On the Design of R-Based Scalable Frameworks for Data Science Applications

Stefan Peter Theußl

January, 2020

Abstract

This thesis is comprised of three papers *On the Design of R-Based Scalable Frameworks for Data Science Applications*. We discuss the design of conceptual and computational frameworks for the R language for statistical computing and graphics and build software artifacts for two typical data science use cases: optimization problem solving and large scale text analysis. Each part follows a design science approach. We use a verification method for the software frameworks introduced, i.e., prototypical instantiations of the designed artifacts are evaluated on the basis of real-world applications in mixed integer optimization (consensus journal ranking) and text mining (culturomics).

The first paper introduces an extensible object oriented R Optimization Infrastructure (**ROI**). Methods from the field of optimization play an important role in many techniques routinely used in statistics, machine learning and data science. Often, implementations of these methods rely on highly specialized optimization algorithms, designed to be only applicable within a specific application. However, in many instances recent advances, in particular in the field of convex optimization, make it possible to conveniently and straightforwardly use modern solvers instead with the advantage of enabling broader usage scenarios and thus promoting reusability. With **ROI** one can formulate and solve optimization problems in a consistent way. It is capable of modeling linear, quadratic, conic, and general nonlinear optimization problems. Furthermore, the paper discusses how extension packages can add additional optimization solvers, read/write functions and additional resources such as model collections. Selected examples from the field of statistics conclude the paper.

With the second paper we aim to answer two questions. Firstly, it addresses the issue on how to construct suitable aggregates of individual journal rankings, using an optimization-based consensus ranking approach. Secondly, the presented application serves as an evaluation of the **ROI** prototype. Regarding the first research question we apply the proposed method to a subset of marketing-related journals from a list of collected journal rankings. Next, the paper studies the stability of the derived consensus solution, and degeneration effects that occur when excluding journals and/or rankings. Finally, we investigate the similarities/dissimilarities of

the consensus with a naive meta-ranking and with individual rankings. The results show that, even though journals are not uniformly ranked, one may derive a consensus ranking with considerably high agreement with the individual rankings.

In the third paper we examine how we can extend the text mining package **tm** to handle large (text) corpora. This enables statisticians to answer many interesting research questions via statistical analysis or modeling of data sets that cannot be analyzed easily otherwise, e.g., due to software or hardware induced data size limitations. Adequate programming models like MapReduce facilitate parallelization of text mining tasks and allow for processing large data sets by using a distributed file system possibly spanning over several machines, e.g., in a cluster of workstations. The paper presents a plug-in package to **tm** called **tm.plugin.dc** implementing a distributed corpus class which can take advantage of the **Hadoop** MapReduce library for large scale text mining tasks. We evaluate the presented prototype on the basis of an application in culturomics and show that it can handle data sets of significant size efficiently.

Keywords: binary optimization, consensus ranking, convex programming, culturomics, distributed computing, **Hadoop**, high performance computing, journal ranking, linear programming, MapReduce, mathematical programming, meta-ranking, mixed integer programming, nonlinear programming, optimization, parallel computing, quadratic programming, R, software design, text mining.

Kurzfassung

Die vorliegende Dissertation *On the Design of R-Based Scalable Frameworks for Data Science Applications* besteht aus drei wissenschaftlichen Artikeln. Wir diskutieren das Design von konzeptionellen und komputationalen Frameworks für R, eine Sprache für statistisches Rechnen und Grafiken und entwerfen Software-Artefakte für zwei typische Anwendungsfälle von Data Science: das Lösen von Optimierungsproblemen sowie rechen- und datenintensive Textanalyse. Jeder Teil folgt einem Design-wissenschaftlichen Ansatz. Wir verwenden eine Verifikationsmethode für die vorgestellten Software-Frameworks, d.h. prototypische Instanziierungen der entworfenen Artefakte werden auf der Grundlage von realen Anwendungen in der Gemischt-Ganzzahligen-Optimierung (Consensus Journal Ranking) und im Text-Mining (Culturomics) bewertet.

Der erste Artikel stellt eine erweiterbare objektorientierte R Optimierungsinfrastruktur (**ROI**) zur Förderung einerseits der Nutzung von Optimierung in R und andererseits von R als Werkzeug zur Optimierung vor. Entsprechende Methoden spielen in vielen Bereichen der Statistik, des maschinellen Lernens und in der Data Science eine wesentliche Rolle. Häufig sind die Implementierungen dieser Methoden auf hochspezialisierte Optimierungsalgorithmen angewiesen, die so konzipiert sind, dass sie nur in dem für sie bestimmten Anwendungsfall anwendbar sind. Die jüngsten Fortschritte, insbesondere im Bereich der konvexen Optimierung, ermöglichen es jedoch in vielen Fällen, moderne Löser komfortabel und unkompliziert einzusetzen, mit dem Vorteil, breitere Nutzungsszenarien zu ermöglichen und damit die Wiederverwendbarkeit zu fördern. **ROI** ist in der Lage, lineare, quadratische, konische und allgemeine nichtlineare Optimierungsprobleme in einer konsistenten Art und Weise zu modellieren. Darüber hinaus verwaltet die Infrastruktur zahlreiche verschiedene Löser, Umformulierungen, Problemsammlungen sowie Funktionen zum Lesen und Schreiben von Optimierungsproblemen in verschiedenen Formaten.

Mit dem zweiten Beitrag wollen wir zwei Fragen beantworten. Erstens, geht es um die Frage, wie man ein geeignetes Aggregat von einzelnen Rankings wissenschaftlicher Publikationsformate (Fachzeitschriften) unter Verwendung eines optimierungsbasierten Konsens-Ranking-Ansatzes konstruiert. Zweitens dient die vorgestellte Anwendung als eine Art Evaluierung des **ROI** Prototypen. Was die er-

ste Forschungsfrage betrifft, so wenden wir die vorgeschlagene Methode auf eine Teilmenge von marketingbezogenen Zeitschriften aus einer Liste von gesammelten Zeitschriftenrankings an. Wir untersuchen auch die Stabilität der abgeleiteten Konsensuslösung und die Degenerationseffekte durch den Ausschluss von Zeitschriften und/oder Rankings. Schließlich untersuchen wir die Ähnlichkeiten/Differenzen des Konsenses zu einem naiven Meta-Ranking und den individuellen Rankings. Die Ergebnisse zeigen, dass es möglich ist, auch wenn Zeitschriften nicht einheitlich gereiht sind, ein Konsensus-Ranking mit sehr hoher Übereinstimmung zu den einzelnen Rankings abzuleiten.

Im dritten Beitrag untersuchen wir, wie wir das Text-Mining-Paket **tm** für die effiziente Verarbeitung großer (Text-)Korpora erweitern können. Auf diese Weise können Statistikerinnen und Statistiker viele interessante Forschungsfragen durch statistische Analyse oder Modellierung von Datensätzen beantworten, die sonst nicht einfach zu analysieren sind, z.B. aufgrund von software- oder hardwarebedingten Datengrößenbeschränkungen. Entsprechende Programmiermodelle wie MapReduce erleichtern die Parallelisierung von Text-Mining-Aufgaben und ermöglichen die Verarbeitung großer Datensätze unter Verwendung eines verteilten Dateisystems, das sich gegebenenfalls über mehrere Maschinen erstreckt, z.B. in einem Cluster von Workstations. Der Artikel stellt ein Plug-in-Paket für **tm** namens **tm.plugin.dc** vor, das eine neue Klasse Distributed Corpus implementiert, die die Vorteile der **Hadoop** MapReduce-Bibliothek für groß angelegte Text-Mining-Aufgaben nutzen kann. Wir evaluieren den vorgestellten Prototyp anhand einer kulturwissenschaftlichen Anwendung (Culturomics) und zeigen, dass er mit Datensätzen von erheblicher Größe effizient umgehen kann.

Schlagwörter: Binäre Programmierung, Culturomics, Gemischt-Ganzzahlige Optimierung, **Hadoop**, Hochleistungsrechnen, Konsensusranking, Konvexe Programmierung, Lineare Programmierung, MapReduce, Mathematische Programmierung, Meta-Ranking, Nichtlineare Programmierung, Optimierung, Parallelisierung, Quadratische Programmierung, R, Software Design, Text-Mining, Verteiltes Rechnen, Zeitschriftenrankings.

Acknowledgments

First and foremost I would like to thank my main guides of this journey, my supervisors Kurt Hornik and David Meyer, for their mentorship, for many fruitful discussions and for their great feedback. It was an honor for me to walk this part of my life with them.

Another guide significantly shaping this journey was Diethelm Würtz with whom I had many interesting discussions and productive cooperations on the topic of optimization, which has become an integral part of this dissertation. It is very unfortunate that he could not live to see the end of this journey.

Then I would like to thank my co-authors of the presented papers in this dissertation, Ingo Feinerer and Thomas Reutterer, for their collaboration and guidance while working on these joint research projects. Special thanks goes to Florian Schwendinger, with whom I made the last part of this journey, not only for being a great research fellow but also for being a motivating sparring partner on many of my R coding endeavors. With this in mind, some coding projects would not have been that successful without the help of Christian Buchta, with whom I shared the interest in many topics around programming and technology while being at the institute. Thank you. A major role in shaping the early stages of this path played Bettina Grün and Achim Zeileis. I would like to thank them for always having an open ear and by supporting me with great feedback. Furthermore, many thanks go to Josef Leydold, Klaus Pötzelberger, and others who have been always around at my time at the institute and with whom I had many on- and off-research discussions and joint lunch breaks. Thanks are due also to Uwe Ligges and Ronald Hochreiter, for being co-authors, discussion partners and most importantly inspirers and motivators.

I would like to thank my kids Amelie and Elias, who are my sunshine and always put a smile on my face, especially when things have been cloudy and did not work out as expected. They gave me an extra motivation to complete this project since they often had to do without me at that time. Thanks are due also to my parents for their ongoing encouragement and who were always open to welcome me in the green and beautiful Styria to take a break from the stressful everyday life. Last but not least, I would like to thank my wife Selma for proofreading parts of this dissertation but even more for her patience and support while me being on this journey.

Contents

List of Figures

List of Tables

Chapter 1

General Introduction

Already over 50 years ago Tukey (1962) pointed out that data analysis involves more than doing statistical inference. It involves everything related to learning from data (Chambers 1993). First labeled by Cleveland (2001), the new field of *data science* emerged, which is a cross-disciplinary field that mainly builds on statistics and computer science, but also on many others like communication, management, economics, or sociology. It studies the methods involved in the analysis and processing of data and proposes technology to improve methods in an evidence-based manner (Donoho 2017).

Data science processes are typically supported by appropriate information systems employing computational methods to analyze data (of considerable size) using mathematical or statistical models. In this thesis we discuss the design of conceptual and computational frameworks for the R language for statistical computing and graphics (R Core Team 2019a) and build software artifacts for two typical data science use cases: optimization problem solving and large scale text analysis. We use a verification method for the software frameworks introduced, i.e., prototypical instantiations of the designed artifacts are evaluated on the basis of real-world applications (consensus journal ranking and culturomics). In this sense we follow a *design science* approach as elaborated in Hevner, March, Park, and Ram (2004).

Our main findings have been published in the Journal of Statistical Software and the Journal of Business Research, respectively. The research work is contained in three papers, of which this dissertation is composed.

The remainder of this chapter is organized as follows. In Section 1.1 we provide the motivation as well as the purpose and scope of the conducted research. The research design that was employed is discussed in Section 1.2. In Section 1.3 a description of the types of artifacts we develop is given followed by a description of the evaluation process (Section 1.4). At the end of this chapter, in Section 1.5, we outline how the remainder of this dissertation is structured.

1.1 Overview of research work and objectives

This section gives an overview of the research work contained in this dissertation. It provides the motivation as well as the purpose and scope of the artifacts to be developed for each of the two data science use cases: optimization problem solving (the first two papers, Chapters 2 and 3 of this dissertation) and large scale text analysis (the third paper, Chapter 4).

1.1.1 Optimization problem solving

Chapter 2: ROI: an Extensible R Optimization Infrastructure

Optimization plays an important role in many methods routinely used in statistics, machine learning and data science. Often, implementations of these methods rely on highly specialized optimization algorithms, designed to be only applicable within a specific application. However, in many instances recent advances, in particular in the field of convex optimization, make it possible to conveniently and straightforwardly use modern solvers instead with the advantage of enabling broader usage scenarios and thus promoting reusability. The paper introduces an R Optimization Infrastructure package (**ROI**) which provides an extensible infrastructure to model linear, quadratic, conic and general nonlinear optimization problems in a consistent way. Furthermore, the infrastructure administers many different solvers, reformulations, problem collections and functions to read and write optimization problems in various formats.

Motivation

Optimization is the process of allocating scarce resources to a feasible set of alternative solutions in order to minimize (or maximize) the overall outcome. Given a function $f_0 : \mathbb{R}^n \to \mathbb{R}$ and a set $\mathcal{C} \subseteq \mathbb{R}^n$ we are interested in finding an $x^* \in \mathbb{R}^n$ that solves

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & x \in \mathcal{C}. \end{aligned} \tag{1.1}$$

The function f_0 is called the *objective function*. A point x is said to be feasible if it satisfies every constraint given by the set $\mathcal{C}$ of all feasible points defining the *feasible region*. If $\mathcal{C}$ is empty, then we say that the optimization problem (OP) is *infeasible*. Since maximization problems can be expressed as minimization problems by just changing the sign in the objective function, we subsequently will mainly deal with minimization problems.

An OP can be *bounded* or *unbounded*. In the latter case, there are sequences

$x^j \in \mathcal{C}$ for which the value of the objective tends to $-\infty$ in a minimization problem, symbolically $f_0(x^j) \to -\infty$ as $j \to +\infty$. Thus, a problem like in Equation 1.1 may or may not have a *solution*. If the problem is neither infeasible nor unbounded then we can often find a vector $x^* \in \mathcal{C}$ that satisfies

$$f_0(x^*) \leq f_0(x), \ \forall x \in \mathcal{C},$$

which is commonly referred to as a solution of the OP.

Since any feasible set $\mathcal{C}$ can be expressed by the combination of constraint functions, the OP from Equation 1.1 can be written as:

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq b_i, \quad i = 1, \ldots, m, \end{array} \tag{1.2}$$

where $b \in \mathbb{R}^m$ is the so-called right-hand-side. The constraints $f_i, \ i = 1, \ldots, m$ are sometimes referred to as *functional constraints* (Ben-Tal and Nemirovski 2001; Nesterov 2004). Since any equality constraint can be expressed by two inequality constraints and vice versa any inequality constraint can be expressed as an equality constraint by adding additional variables (also called *slack variables*), it is common practice to define OPs only in terms of either equality, less than or equal or greater than or equal constraints, to avoid redundancies.

Based on the functional form of the objective function and of the constraints, OPs can be divided into linear and nonlinear OPs (and sub-classes thereof). In this dissertation we distinguish between the following problem classes: linear programming, quadratic programming, conic programming and general nonlinear programming. Furthermore, additional requirements are added to the optimization problem if some of the objective variables can only be of type integer. Considering Equation 1.2, a problem is called a mixed integer problem (MIP) if the (type) constraint $x_k \in \mathbb{Z}$ for at least one k is added.

Optimization in R For R, being a general-purpose tool for scientific computing and data science, optimization and access to highly efficient solvers play an important role. The field of optimization already has many resources to offer, like software for modeling, solving and randomly generating optimization problems, as well as optimization problem collections used to benchmark optimization solvers. In order to exploit the available resources more conveniently, over the years many modeling tools have emerged. One of the first systems used to model linear optimization problems is the so-called Mathematical Programming System (`MPS`) format (see Kallrath 2004). Developed in the 1960's, the `MPS` format today seems rather archaic but it is still widely used to store and exchange linear problems and is

supported by most of the linear optimization solvers. Later, algebraic modeling languages (AMLs) (e.g., GAMS (Bisschop and Meeraus 1982) and AMPL (Fourer, Gay, and Kernighan 1989)) became available. AMLs are domain specific languages (DSLs) dedicated to optimization. Today modern optimization systems are typically implemented in high-level programming languages like Julia (Bezanson, Edelman, Karpinski, and Shah 2017), MATLAB (The MathWorks Inc. 2017), Python (Python Software Foundation 2017) or R.

Despite R having access to many modern optimization solvers which are capable of solving a wide class of optimization problems (see, e.g., the CRAN optimization and mathematical programming task view by Theußl, Borchers, and Schwendinger 2019a), it is still commonplace to develop highly sophisticated special purpose code (SPC) for many statistical problems. The reasons are many. To name but a few: 1) *availability*, i.e., many solvers have not been easily available in R, 2) *capability*, i.e., problems could not be solved due to a lack of adequate solvers, and 3) *efficiency*, i.e., SPC tends to be faster.

Scope of the study and research questions

At first we survey available R packages concerned with solving problem classes as introduced above. We then closely look at their commonalities and differences and how optimization problems are to be formulated and solved using the respective solver. This leads to our main research question of the paper.

Research question: How can we design a consistent framework for constructing and solving optimization problems of different types providing a unified interface to available solvers as well as a modeling mechanism borrowing its strength from the rich language features R has to offer?

The main part of the paper then aims to elaborate the conceptual design of **ROI** and how to use it to formulate and solve optimization problems. A proper implementation of this design makes it more attractive to add new solvers to the R solver landscape, e.g., to take advantage of recent advances in conic optimization (increase availability). Furthermore, we discuss how extension packages can add additional optimization solvers, read/write functions and additional resources (increase capability). Furthermore, allowing package developers to *plug-in* new solvers quite effortlessly not only makes it easy to use their highly efficient code for a given problem but possibly also in many other applications (eliminate efficiency detriments). Selected examples from the field of statistics will conclude the paper.

Chapter 3: How to Derive Consensus Among Various Marketing Journal Rankings?

Despite the increasing popularity of journal rankings to evaluate the quality of research contributions, the individual rankings for journals that ranked below the top-tier of publications usually feature only modest agreement. Attempts to merge rankings into meta-rankings suffer from some methodological issues, such as mixed measurement scales and incomplete data. The paper addresses the issue of how to construct suitable aggregates of individual journal rankings, using an optimization-based consensus ranking approach. We apply the proposed method to a subset of marketing-related journals from a list of collected journal rankings. Next, the paper studies the stability of the derived consensus solution, and degeneration effects that occur when excluding journals and/or rankings. Finally, we investigate the similarities/dissimilarities of the consensus with a naive meta-ranking and with individual rankings. The results show that, even though journals are not uniformly ranked, one may derive a consensus ranking with considerably high agreement with the individual rankings.

Motivation

There are two major challenges in aggregating journal ranking data sets, which have not yet been adequately resolved by existing approaches: (1) The different measurement scales used by the rankings and (2) incomplete information (Franke and Schreier 2008; Mingers and Harzing 2007; Schrader and Hennig-Thurau 2009). The first issue refers to the fact that the available rankings make use of quite different scale levels including binary (yes/no), ordinal (e.g., by assigning grades A+, A, B, etc.), or numeric scores (e.g., impact factors above 0) to construct their rankings. This issue makes the aggregation of rankings with conventional statistical methods cumbersome. Even more problematic appears to be the second issue, which is related to the typically large amount of "missing observations" in ranking data sets. They accrue because the various rankings cover only subsets of journals, which usually coincide only partially. Thus, the sparsity of the data set is generally increasing with broader rankings.

Hornik and Meyer (2007) have shown that given the above constraints it is nevertheless possible to derive meta-rankings of journals by solving consensus optimization problems.

Scope of the study and research questions

The paper has a clear application oriented focus since the implementation of the presented methodology serves as the evaluation of the prototype introduced in

Chapter 2. In doing so we answer the following research question regarding the aggregation of marketing-related journal rankings.

Research question: How can we derive a consensus ranking methodology which optimally synthesizes the individual rankings, i.e., derives a ranking that shows a considerable high level of agreement with the individual rankings?

To answer this question we pursue the following research plan. At first we introduce the journal ranking data used in the study and the challenge of suitably aggregating these rankings. We proceed with the cornerstones of the proposed consensus ranking methodology and provide pointers for implementing the procedures involved. Here, the software artifacts introduced in Chapter 2 play an important role in supporting the research process. Subsequently, we apply the method using a subset of marketing-related journals provided by the *Harzing Journal Ranking Repository* (`http://www.harzing.com/jql.htm`). We also explore the sensitivity of the consensus ranking methodology to variations in the number of rankings and/or journals by studying stability and degeneration issues. In addition, we compare it to a naive ranking derived by a simple averaging of ranks. Finally, the paper concludes with a discussion on the value of consensus rankings and outline some points for future research.

1.1.2 Large scale text analysis

Chapter 4: A tm Plug-In for Distributed Text Mining in R

R has gained explicit text mining support with the **tm** package enabling statisticians to answer many interesting research questions via statistical analysis or modeling of (text) corpora. However, we typically face two challenges when analyzing large corpora: (1) the amount of data to be processed in a single machine is usually limited by the available main memory (i.e., RAM), and (2) the more data to be analyzed the higher the need for efficient procedures for calculating valuable results. Fortunately, adequate programming models like MapReduce facilitate parallelization of text mining tasks and allow for processing data sets beyond what would fit into memory by using a distributed file system possibly spanning over several machines, e.g., in a cluster of workstations. In the paper we present a plug-in package to **tm** called **tm.plugin.dc** implementing a distributed corpus class which can take advantage of the **Hadoop** MapReduce library for large scale text mining tasks. We show on the basis of an application in culturomics that we can efficiently handle data sets of significant size.

Motivation

The **tm** package (Feinerer 2018), originally presented in Feinerer, Hornik, and Meyer (2008), provides sophisticated methods for document handling, transformations, filters, and data export (such as constructing document-term matrices). With a focus on extensibility based on generic functions and object-oriented inheritance, **tm** makes it possible to apply a multitude of existing methods in the R world to text data structures as well.

However, the endeavor to analyze huge text corpora using **tm** is the source of two challenges: (1) the amount of data to be processed in a single machine is usually limited by the available main memory (i.e., RAM), and (2) an increase of the amount of data to be analyzed leads to higher demand for efficient procedures for calculating valuable results. Thus, it is highly imperative to find a solution which overcomes the memory limitation (e.g., by splitting the data into several pieces) and to markedly reduce the run-time by distributing the workload across available computing resources (such as CPU cores or virtual machine instances). Typically, we consider distributed memory platforms like clusters of workstations for such applications since they are scalable in terms of CPUs and memory (disk space and RAM) employed. Furthermore, many different programming models and libraries like the message passing interface (MPI) facilitate working with this kind of *high performance computing* systems. Many of those libraries can directly be employed in R (see Schmidberger, Morgan, Eddelbuettel, Yu, Tierney, and Mansmann 2009, for further references). Still, one open question remains: is there an efficient way to handle large corpora using R?

Scope of the study and research questions

The first step is to review **tm** as well as the typical workflow when using the package. We expect to face two challenges which need to be tackled when working with large data sets. Firstly, big data sets, i.e., data sets which do not fit into main memory like corpora with several millions of documents, cannot easily be constructed and thus processed with the basic facilities provided by **tm**. Secondly, iterations over several millions of documents are rather time consuming. For example performing typical preprocessing steps like stemming or stop word removal on raw text documents can become quite expensive in terms of computing time when the corpus is very large.

Fortunately, operations such as applying transformations and filters are highly amenable to parallelization by construction, as they can separately be applied to each document without side effects. Furthermore, another concept in **tm** named *sources* is used to abstract document acquisitions. Although we use different sophisticated mechanisms for corpus construction like using database back ends it is

conceptually appealing and possible to allocate the storage in a distributed manner since communication is usually not limited by a single bottleneck. Ideally, even subsets of the original data set (the corpus) are stored physically distributed on several machines (e.g., in a cluster of workstations). This will not only allow us to increase storage space for data (scaling with the number of participating machines) but also reduce communication costs for parallel computation since only those documents stored locally on a given machine are to be processed on the respective system. Thus, we can use these two approaches (parallel processing, distributing data) to tackle the challenges indicated above. This leads to our main research question of the paper.

Research question: How can we design an extension to **tm** such that we are able to transparently distribute the documents on one or several storage entities, apply functions on the subsetted corpus possibly in parallel, and gather results on a cluster of workstations or other (distributed) computing platforms?

In the following step we show that both requirements (parallel processing, distributing data) are often fulfilled by well-established distributed programming models such as MapReduce (Dean and Ghemawat 2008). Typically, MapReduce is used in combination with another important building block: the distributed file system (DFS, Ghemawat, Gobioff, and Leung 2003). This approach readily enables and takes care of data distribution and suitable parallel processing of parts of the data in a functional programming style (Lämmel 2007). Given that the MapReduce model fits to the workflow presented above and corresponding open source software libraries are available, it seems an excellent choice when we need to process large corpora in text mining scenarios.

After that we discuss the design of the package **tm.plugin.dc** building on functionality provided by interfaces to **tm** and to MapReduce environments. We show that by selecting appropriate building blocks we are not only able to employ the tools provided by Hadoop but also any abstract registered (distributed) storage and parallel computing environment. Furthermore, we provide a new *distributed corpus* class along with corresponding methods which allow us to analyze large corpora seamlessly without knowing how to use the underlying components of MapReduce or other libraries.

1.2 Research design

Design science research typically involves the creation of an *artifact* and/or design theory as a means to improve the current state of practice as well as existing research knowledge (Vaishnavi, Kuechler, and Petter 2004/19; Baskerville, Baiyere,

Gregor, Hevner, and Rossi 2018). When focusing on the development of artifacts, then, according to Vaishnavi *et al.* (2004/19), design science research involves two primary activities to improve and understand the behavior of aspects of information systems: the creation of new knowledge through design of novel or innovative artifacts and the analysis of the artifact's use and/or performance with reflection and abstraction. According to March and Smith (1995) the software artifacts designed for that purpose can be of type *constructs* (vocabulary and symbols), *models* (abstraction and representations), *methods* (algorithms and practices), and *implementations/instantiations* (implemented and prototype systems).

In the data science context, this could mean creating artifacts which allow us to construct and solve optimization problems of different types providing a modeling mechanism borrowing its strength from the rich language features R has to offer, or, which are capable of extracting data from previously unanalyzable data sets (e.g., due to its large size) and/or creating innovations which improve the time required to analyze such data sets. Knowledge and understanding of the problem and its solution are achieved in building and application of the designed artifacts.

By adopting a proper design science research methodology this allows us to answer the research questions stated in Sections 1.1.1 and 1.1.2 and contribute to the knowledge base of how to design R-based scalable frameworks for data science applications.

1.2.1 Methodology

Hevner *et al.* (2004) introduced seven guidelines for understanding, executing and evaluating design science research. It requires (1) the creation of an innovative, purposeful artifact; (2) for a specific problem domain; (3) yielding utility for the specified problem, which is thoroughly evaluated; (4) solving an heretofore unsolved problem or a known problem in a more efficient manner, (5) the artifact being rigorously defined, formally represented, coherent, and internally consistent; (6) the artifact (or the process creating it) incorporating or enabling a search process to find an effective solution; and (7) effectively communicating the results of the design science research.

In this thesis we follow these guidelines to conduct effective design science research, i.e., we thoroughly define, build and evaluate a prototypical implementation in order to solve the research questions at hand. In three published papers we communicate our findings to the research community.

1.2.2 Application of the research design

Our chosen design search process is a two step approach (guideline 6). As a first step we conducted a literature review on the problem domain of interest: optimization problem solving and large scale text analysis as data science use cases on which we want to work with the R language for statistical computing and graphics. This enables us to get an overview on the topic and to identify research gaps (guideline 2).

After formalizing the building blocks that make up the artifact, the derived design is instantiated in the form of an early prototype (e.g., an R package) that is evaluated using standard test libraries like the MIPLIB (Koch, Achterberg, Andersen, Bastert, Berthold, Bixby, Danna, Gamrath, Gleixner, Heinz, Lodi, Mittelmann, Ralphs, Salvagnin, Steffy, and Wolter 2011) (containing academic and industrial MILP applications) or standard text collections like the *Reuters-21578* corpus (Lewis 1997), respectively. Once we have proven the validity of the design, e.g., by comparing the output of our instantiations with that of similar artifacts (guidelines 3 and 5), we publish the early version of the package on CRAN[1] (`https://CRAN.R-project.org`), the standard repository for R packages, and go to step two.

In the second design step we start with the reflection of the previous design step's results and refine our design accordingly. In addition we incorporate feedback we have received through the presentation of the early instantiations at international conferences covering topics ranging from domain-specific such as the Computational Management Science, the Operations Research (OR, organized by Gesellschaft für Operations Research e.V., `https://www.gor-ev.de/`), or high performance and parallel computing (e.g., Euro-Par, `https://euro-par.org`) to general R-related topics such as the R/Finance (`https://www.rinfinance.com/`), or the useR*!* (`https://www.r-project.org/conferences/`) as well as the feedback we have received from early users of the published packages. We use this feedback and additional theoretical knowledge to adapt our design. Subsequently we develop an improved artifact based on the adapted design and conduct a second evaluation.

The final instantiation of the prototype then solves the data science use case: optimization problem solving (consensus journal ranking) and large scale text analysis (culturomics), respectively. According to Gregor and Hevner (2013) the final design can be classified as an improvement, a new solution for known problems (guidelines 1 and 4). The results are published in a renowned journal of the field and the artifacts are made available as open source software on CRAN (guideline 7).

[1] An R package published on CRAN fulfills certain formal requirements (see the policies available at `https://cran.r-project.org/web/packages/policies.html`) in addition to the ones outlined in R Core Team (2019b). Both, an artifact being a properly built R package and passing the CRAN acceptance checks, can be seen as another test that the artifact "works" and is valid.

1.3 Artifact description

In this section we describe the most important building blocks for implementing the prototypes for the data science use cases. The conceptual framework (including abstractions, constructs, and methods) and the implementation (instantiations) are described for each of the two innovations in data science: *A general optimization infrastructure for R* (optimization problem solving) and *A **tm** plug-in for distributed text mining in R* (large scale text analysis).

1.3.1 A general optimization infrastructure for R

The basic requirements for a general optimization infrastructure for R in terms of the conceptual framework and the implementation are as follows.

Constructs

The main constructs of a general optimization infrastructure are the *optimization problem* and its *solution*. Both constructs are implemented as S3 classes.

Optimization problem Based on the review in Sections 1.1.1 it seems natural to instantiate OPs based on an *objective* function, one or several *constraints*, *types* and *bounds* of the objective variables, as well as the direction of optimization (whether a minimum or a *maximum* is sought). The elements of this class are constructs on their own and possibly also implemented as S3 classes. Based on the functional form of the objective function and of the constraints as well as the types of the variables the problem class of the respective optimization problem is implicitly given (i.e, being either a linear programming, quadratic programming, conic programming, or general nonlinear programming problem; being an MIP or not).

Solution The solution class requires information about the solution, the objective value at optimum, and the solving process (at least the information if it was successful or not).

Functions and methods

We use generic functions from the R language and define methods to compute on objects of the defined classes (and sub-classes thereof). We add new generics where appropriate. We define functions to construct (components of) the OP, extract information from the OP and the solution object, and interact with the object.

The main computation on OPs is to *solve* it, i.e., to find a solution of the given problem.

Solving optimization problems The function to solve an optimization problem typically considers an object containing the formulation of the OP of the class described above, the reference to the solver to be used, and possibly further information needed by the solver to properly interact with the OP.

Implementation

The artifact is implemented as an R package since this is the standard and proper way of extending R (see R Core Team 2019b, for details on writing R extensions). The package is called **ROI**. It implements all constructs, functions, and methods described above.

As indicated in Section 1.1.1 a proper implementation of this design makes it more attractive to add new solvers to the R solver landscape, or to add additional optimization solvers, read/write functions and additional resources. Thus, the artifact is required to be modular allowing developers to *plug-in* new solvers or add resources quite effortlessly. This is achieved by the artifact additionally providing all the necessary classes and methods and managing the extensions, which in turn are R packages.

1.3.2 A tm plug-in for distributed text mining in R

The basic requirements for a **tm** plug-in for distributed text mining in R in terms of the conceptual framework and the implementation are as follows.

Abstractions

The bridge between MapReduce libraries and the **tm** infrastructure is characterized by two main design concepts: *distributed storage* and *parallel computation*. The one side of the bridge is designed in such a way that it provides a corpus implementation which can transparently be used in combination with the existing **tm** infrastructure. Via the other side of the bridge we can access and modify data stored on a (distributed) file system (e.g., the **Hadoop** Distributed File System, HDFS).

Constructs

The main construct of a scalable text mining framework extending **tm** to focus on is the *corpus* implemented as an S3 class. The derived construct is the *distributed corpus*.

Corpus In **tm** the main data structure is a *corpus*, an entity similar to a database holding text documents in a generic way. It can be seen as a container to store

a collection of text documents where additional metadata is provided on both the corpus (e.g., date of creation, creator, etc.) and document level (e.g., annotations, authors, language, etc.).

Distributed corpus Since **tm** corpora are lists of objects of a class describing text documents enriched with metadata it seems only natural to encapsulate this storage abstraction ensuring that the documents are distributed on one or several storage entities instead of being held in memory.

Functions and methods

Distributed storage Appropriate methods for *distributed corpus* objects ensure that the local files delivered by a source instance are transparently loaded into the distributed storage and retrieved from the storage when needed.

Parallel computation Several methods have been implemented to abstract the process of document manipulation, like transformations and filtering, and data export, like the construction of a so-called *document-term matrix* (DTM) holding frequencies of distinct terms for each document. Once we have documents stored on the distributed storage, we want to perform computations on the data pieces local to each processing node. Such computations are highly parallel and scale with the number of available workstations. Conceptually, for transformations or filtering we would apply a `map` function (similar to the one known from functional programming) to each element of the corpus. Other typical operations on distributed data are *collective* operations. Functions of this type commonly gather or aggregate data based on a given set of instructions, like `reduce`. Using (a combination of) `map` and `reduce` operations allows us to transform documents and/or export other constructs like DTMs in a very efficient, parallel, manner.

Implementation

Similar to the general optimization infrastructure the artifact for the use case large scale text analysis is implemented as an R package. It is called **tm.plugin.dc** and implements all constructs, functions, and methods described above.

1.4 Evaluation

We follow the epistemological approach as introduced in Section 1.2. According to Gregor and Hevner (2013) the artifacts can be evaluated in terms of the following criteria: validity, utility, quality, and efficacy.

As indicated in Section 1.2.2, the design search process is defined as iterative, i.e., the instantiated prototype of the derived design is evaluated by simply showing that it does what it is meant to do. For both of our data science use cases we use tests with suitable standard test data sets to continuously improve the artifact and test instantiations thereof. We prove the validity of the design, e.g., by comparing the output of our instantiations with that of similar artifacts.

The final utility and efficacy of the prototypical instantiation of the designed artifact, i.e., packages **ROI** and **tm.plugin.dc**, is evaluated on the basis of real world applications of the respective data science use cases (consensus journal ranking and culturomics). This proof of concept answers the research questions formulated in Section 1.1. Results of a benchmark experiment complete the analysis.

In addition, we can measure the utility of the developed R packages outside the individual data science use cases, i.e., the development environment, by looking at other community contributions when they directly or indirectly reuse code of the respective package (Theußl, Ligges, and Hornik 2011). A higher number of references to the package would indicate a higher utility for other use cases.

1.4.1 Standard test data sets

Test problems for ROI

Test problem collections are commonly used in optimization to evaluate and compare the performance of solvers. As each class of optimization problems has its own test sets stored in various formats, **ROI** currently provides access to NETLIB-LP, MIPLIB and the **globalOptTests** package. **ROI** makes these data collections (test problem sets) available in a common format, so users can easily compare the different solvers and developers interested in creating optimization software can use them to test their packages.

NETLIB-LP The NETLIB-LP (Gay 1985) is a collection of linear programming problems, which, even though the main part was created more than 30 years ago is still used today.

MIPLIB Mixed integer optimization problems are commonly evaluated using MIPLIB (Koch *et al.* 2011), an extensive collection of academic and industrial MILP applications.

globalOptTests The **globalOptTests** (Mullen 2014a) package contains 50 box constrained nonlinear global OPs for benchmarking purposes.

Test problems for tm.plugin.dc

Reuters-21578 The Reuters-21578 data set (Lewis 1997) contains stories collected by the Reuters news agency. The data set is publicly available and has been widely used in text mining research within the last decade. It contains 21,578 short to medium length documents in XML format (obtainable e.g., from http://ronaldo.cs.tcd.ie/esslli07/data/) covering a broad range of topics, like mergers and acquisitions, finance, or politics. To download the corpus use:

NSF Research Awards Abstracts This data set consists of 129,000 plain text abstracts describing NSF awards for basic research submitted between 1990 and 2003. The data set can be obtained from the *UCI Machine Learning Repository* (http://archive.ics.uci.edu/ml/). The corpus is divided into three parts. We used the largest part (*Part 1*) in our experiments.

Reuters Corpus Volume 1 Lewis, Yang, Rose, and Li (2004) introduced the RCV1 consisting of about 800,000 (XML format) documents as a test collection for text categorization research. The documents contained in this corpus were sent over the Reuters Newswire (https://www.reutersagency.com/en/products/newswires/) during a 1-year period between 1996-08-20 and 1997-08-19. RCV1 covers a wide range of international topics, including business & finance, lifestyle, politics, sports, etc. The stories were manually categorized in three category sets: topic, industry and region.

NYT Annotated Corpus The largest data set in our experiment contains over 1.8 million articles published by the New York Times between 1987-01-01 and 2007-06-19 (Sandhaus 2008). Documents and corresponding metadata are provided in an XML like format: News Industry Text Format (NITF).

1.4.2 Use case: optimization problem solving

A prototypical instantiation of the designed artifact, i.e., package **ROI**, is evaluated on the basis of real-world applications. One set of applications is dedicated to solving statistical inference problems which goes hand in hand with solving optimization problems (OPs). As such statisticians, data scientists, and others who regularly employ computational methods ranging from various types of regression (e.g., constrained least squares, regularized least squares, nonlinear least squares), and classification (e.g., support vector machines, convex clustering) to covariance estimation and low rank approximations (e.g., multidimensional scaling, non-negative matrix factorization) benefit from advances in optimization.

Another set of applications is solving consensus optimization problems. Hornik and Meyer (2007) propose a methodology that obtains consensus rankings from paired comparisons among a set of individual rankings. Accommodating mixed types of measurement scales and being relatively robust for sparse data, this optimization problem can be solved efficiently using any state-of-the-art mixed integer programming solver. The paper *How to Derive Consensus Among Various Marketing Journal Rankings?* answers the second research question formulated in Section 1.1.1.

1.4.3 Use case: large scale text analysis

The prototype **tm.plugin.dc** is evaluated by answering a typical question from the field of culturomics (Michel, Shen, Aiden, Veres, Gray, The Google Books Team, Pickett, Hoiberg, Clancy, Norvig, Orwant, Pinker, Nowak, and Aiden 2011), which deals with the development of human behavior and culture reflected in language and word usage. We investigate how the text coverage in newspaper articles has developed over time. Specifically, we analyze the multitude of text documents published by the New York Times between 1987-01-01 and 2007-06-19. The corresponding corpus consists of 1,855,658 short- to medium-length articles from various genres with a mean of 552 terms per document. As such, this corpus is too large for being handled with the standard text mining tool chain available in R. However, we would be able to use the distributed text mining framework in order to process the corpus quite efficiently making it possible to not only investigate the culturomics question at hand, but also to subject the corpus to a variety of other statistical analyses.

1.5 Structure of this dissertation

The remainder of this dissertation, which is comprised of three papers *On the Design of R-Based Scalable Frameworks for Data Science Applications*, is organized as follows. Chapter 2 (the first paper) introduces an extensible object oriented R Optimization Infrastructure (**ROI**). Then, with Chapter 3 (the second paper), we aim to answer two questions. Firstly, it addresses the issue on how to construct suitable aggregates of individual journal rankings, using an optimization-based consensus ranking approach. Secondly, the presented application serves as an evaluation of the **ROI** prototype. In Chapter 4 (the third paper) we examine how we can extend the text mining package **tm** to handle large (text) corpora. Chapter 5 concludes this dissertation.

Chapter 2

ROI: An Extensible R Optimization Infrastructure

An earlier version of the paper is available online:

Theußl S, Schwendinger F, Hornik K (2019). "ROI: An Extensible R Optimization Infrastructure." *Research Report Series / Department of Statistics and Mathematics 133*, WU Vienna University of Economics and Business, Vienna. URL https://epub.wu.ac.at/5858/.

The paper, with minor changes, was conditionally accepted for publication in the Journal of Statistical Software in March 2019.

2.1 Introduction

Optimization is at the core of inference in modern statistics since solving statistical inference problems goes hand in hand with solving optimization problems (OPs). As such statisticians, data scientists, and others who regularly employ computational methods ranging from various types of regression (e.g., constrained least squares, regularized least squares, nonlinear least squares), and classification (e.g., support vector machines, convex clustering) to covariance estimation and low rank approximations (e.g., multidimensional scaling, non-negative matrix factorization) benefit from advances in optimization, in particular in mixed integer and convex optimization. For example, Bertsimas, King, and Mazumder (2016) show that, thanks to a striking speedup factor of 450 billion in mixed integer optimization in the period of 1991-2015, the NP-hard best subset problem (Miller 2002) can now be solved reasonably fast (number of observations in the 100s and number of variables in the 1000s is solved within minutes). O'Donoghue, Chu, Parikh, and Boyd (2016) introduce the **SCS** solver for convex optimization problems, which can be used to solve among others (logistic) regression with $l_{\{1,2\}}$ regularization, support vector machines, convex clustering, non-negative matrix factorization and graphical lasso.

For R (R Core Team 2019a), being a general-purpose tool for scientific computing and data science, optimization and access to highly efficient solvers play an important role. The field of optimization already has many resources to offer, like software for modeling, solving and randomly generating optimization problems, as well as optimization problem collections used to benchmark optimization solvers. In order to exploit the available resources more conveniently, over the years many modeling tools have emerged. One of the first systems used to model linear optimization problems is the so-called Mathematical Programming System (`MPS`) format (see Kallrath 2004). Developed in the 1960's, the `MPS` format today seems rather archaic but it is still widely used to store and exchange linear problems and is supported by most of the linear optimization solvers. Later, algebraic modeling languages (AMLs) (e.g., GAMS (Bisschop and Meeraus 1982) and AMPL (Fourer *et al.* 1989)) became available. AMLs are domain specific languages (DSLs) dedicated to optimization. Today modern optimization systems are typically implemented in high-level programming languages like Julia (Bezanson *et al.* 2017), MATLAB (The MathWorks Inc. 2017), Python (Python Software Foundation 2017) or R. Among the modern optimization systems, many are DSLs specially suited for convex optimization, such as **YALMIP** (Löfberg 2004) and **CVX** (Grant and Boyd 2014) in MATLAB, **CVXPY** (Diamond and Boyd 2016) and **CVXOPT** (Andersen, Dahl, and Vandenberghe 2019) in Python, **Convex.jl** (Udell, Mohan, Zeng, Hong, Diamond, and Boyd 2014) in Julia and **CVXR** (Fu, Narasimhan, and Boyd 2017) in R. **JuMP** (Lubin and

Dunning 2015) is a DSL implemented in Julia designed for mixed-integer programming. **pyOpt** (Perez, Jansen, and Martins 2012) is a Python package for nonlinear constrained optimization.

Despite R having access to many modern optimization solvers which are capable of solving a wide class of optimization problems (see, e.g., the CRAN optimization and mathematical programming task view by Theußl *et al.* 2019a), it is still commonplace to develop highly sophisticated special purpose code (SPC) for many statistical problems. The reasons are many. To name but a few: 1) *availability*, i.e., many solvers have not been easily available in R, 2) *capability*, i.e., problems could not be solved due to a lack of adequate solvers, and 3) *efficiency*, i.e., SPC tends to be faster.

The paper introduces an extensible object oriented R Optimization Infrastructure (**ROI**) promoting the usage of optimization in R and R as a tool for optimization. In doing so it strives to enable users to formulate problems and experiment with different solvers in a straightforward way, help researchers to find the appropriate solver for their particular problem, or assist package developers to streamline their package dependencies. The framework is composed of package **ROI** (Theußl, Schwendinger, Hornik, and Meyer 2019b) and its (at the time of this writing) 23 companion packages.

In contrast to DSLs, the **ROI** package does not aim to create a new language but provides a modeling mechanism borrowing its strength from the rich language features R has to offer. Optimization problems are constructed in a consistent way and stored in a single object. This makes it possible that problems are easily altered (reused) and shared before they are passed to a unified solve function. Such problems are then formulated and manipulated by using the provided R functions instead of special syntax from DSLs for which highly specialized knowledge would be required. Moreover, we believe that this approach makes it more attractive to add new solvers to the R solver landscape, e.g., to take advantage of recent advances in conic optimization (increase availability). Another key feature of **ROI** is that it is designed to be extensible. Companion packages equip **ROI** with state of the art optimization solvers, benchmark collections and functions to read and write optimization problems in various formats (increase capability). Furthermore, allowing package developers to *plug-in* new solvers quite effortlessly not only makes it easy to use their highly efficient code for a given problem but possibly also in many other applications (eliminate efficiency detriments). Currently **ROI** can be used to model and solve linear, quadratic, second order cone, semidefinite, exponential cone, power cone and general nonlinear optimization problems as well as mixed integer problems. This covers many optimization problems encountered in statistics, machine learning and data science (see, e.g., Koenker and Mizera 2014, for a survey

of convex problems in statistics).

The remainder of the paper is organized as follows: In Section 2.2 we discuss the basic optimization problem classes, with a special focus on the newer developments in convex optimization. A survey of available R packages concerned with solving these problem classes is given in Section 2.3. Sections 2.4 and 2.5 show, respectively, how to formulate and solve optimization problems with the **ROI** package. Based on the tools presented in the previous sections, Section 2.6 provides basic examples. Section 2.7 is dedicated to the extension of **ROI**. Applications in the field of statistics are presented in Section 2.8. Section 2.9 concludes the paper.

2.2 Problem classes

Optimization is the process of allocating scarce resources to a feasible set of alternative solutions in order to minimize (or maximize) the overall outcome. Given a function $f_0 : \mathbb{R}^n \to \mathbb{R}$ and a set $\mathcal{C} \subseteq \mathbb{R}^n$ we are interested in finding an $x^* \in \mathbb{R}^n$ that solves

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & x \in \mathcal{C}. \end{aligned} \tag{2.1}$$

The function f_0 is called the *objective function*. A point x is said to be feasible if it satisfies every constraint given by the set $\mathcal{C}$ of all feasible points defining the *feasible region*. If $\mathcal{C}$ is empty, then we say that the optimization problem is *infeasible*. Since maximization problems can be expressed as minimization problems by just changing the sign in the objective function, we subsequently will mainly deal with minimization problems.

An OP can be *bounded* or *unbounded*. In the latter case, there are sequences $x^j \in \mathcal{C}$ for which the value of the objective tends to $-\infty$ in a minimization problem, symbolically $f_0(x^j) \to -\infty$ as $j \to +\infty$. Thus, a problem like in Equation 2.1 may or may not have a *solution*. If the problem is neither infeasible nor unbounded then we can often find a vector $x^* \in \mathcal{C}$ that satisfies

$$f_0(x^*) \leq f_0(x), \ \forall x \in \mathcal{C},$$

which is commonly referred to as a solution of the OP.

Since any feasible set $\mathcal{C}$ can be expressed by the combination of constraint functions, the OP from Equation 2.1 can be written as:

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \leq b_i, \quad i = 1, \ldots, m, \end{aligned} \tag{2.2}$$

where $b \in \mathbb{R}^m$ is the so-called right-hand-side. The constraints $f_i, \ i = 1, \ldots, m$

are sometimes referred to as *functional constraints* (Ben-Tal and Nemirovski 2001; Nesterov 2004). Since any equality constraint can be expressed by two inequality constraints and vice versa any inequality constraint can be expressed as an equality constraint by adding additional variables (also called *slack variables*), it is common practice to define OPs only in terms of either equality, less than or equal or greater than or equal constraints, to avoid redundancies.

Equation 2.2 is also sometimes referred to as the *primal problem*, which highlights the fact that there exists an alternative problem formulation the *dual problem*. The dual problem is typically defined via the Lagrangian function (Lagrange duality) (Nocedal and Wright 2006).

Several interconnected characteristics exist which determine how efficiently a given OP can be solved, namely convexity, the functional form of the objective, the functional form of the constraints and if the variable x is binary, integer, or continuous. An OP as displayed in Equation 2.1 is convex, if f_0 is convex and the set $\mathcal{C}$ is convex. Whereas modern solvers can efficiently solve a wide range of convex OPs and verify that a *global* solution (i.e., one as good or better than all other feasible solutions) was obtained, the same is mostly not true for non-convex problems (several *local* optima may exist). More information about convex programming can be found in, e.g., Boyd and Vandenberghe (2004); Ben-Tal and Nemirovski (2019).

Based on the functional form of the objective function and of the constraints, OPs can be divided into linear and nonlinear OPs. Furthermore, the class of nonlinear OPs can be further subdivided into conic, quadratic and general nonlinear OPs. In the following we give a formal definition of the different classes of OPs and overview their properties.

2.2.1 Linear programming

A linear program (LP) is an OP where all f_i $(i = 0, \ldots, m)$ Equation 2.2 are linear. Thus an LP can be defined as:

$$\begin{aligned} \text{minimize} \quad & a_0^\top x \\ \text{subject to} \quad & Ax \leq b \end{aligned} \tag{2.3}$$

where x is the vector of objective variables which has to be optimized. The coefficients of the objective function are represented by $a_0 \in \mathbb{R}^n$. $A \in \mathbb{R}^{m \times n}$ is a matrix of coefficients representing the constraints of the LP. Hence, in accordance with Equation 2.2, $Ax \leq b$ could also be written as $a_i^\top x \leq b_i,\ i = 1, \ldots, m$ (here $a_i^\top$ refers to the $i-$th row of the coefficient matrix A). All LPs are convex and usually solved via interior-point or simplex methods. For more information about the origination and mathematical properties of these methods we refer the reader to the book of

Nocedal and Wright (2006).

A typical statistical problem which falls into this problem class is solving the least absolute deviations (LAD) or L_1 regression problem. Following, e.g., Brooks and Dulá (2013) the objective function

$$\text{minimize} \quad \sum_{i}^{n} |y_i - \hat{y}_i|$$

can be expressed as

$$\begin{aligned} \underset{\beta_0,\beta,\mathbf{e}^+,\mathbf{e}^-}{\text{minimize}} \quad & \sum_{i=1}^{n} e_i^+ + e_i^- \\ \text{subject to} \quad & \\ \beta_0 + \beta^\top \mathbf{x}_i + e_i^+ - e_i^- = 0 \qquad & i = 1, \ldots, n \\ \beta_j = -1 \qquad & \\ e_i^+, e_i^- \geq 0 \qquad & i = 1, \ldots, n \end{aligned}$$

given a set of points $\mathbf{x}_i \in \mathbb{R}^m$, $i = 1, \ldots, n$ and the j^{th} column representing the dependent variable.

2.2.2 Quadratic programming

A quadratic program (QP) is a generalization of the standard LP shown in Equation 2.3, where the objective function contains a quadratic part in addition to the linear term. The quadratic part is typically represented by a matrix $Q_0 \in \mathbb{R}^{n \times n}$. Therefore QPs can be expressed in the following manner:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2} x^\top Q_0 x + a_0^\top x \\ \text{subject to} \quad & Ax \leq b. \end{aligned} \tag{2.4}$$

Unlike LPs, not all QPs are convex. A QP is convex if and only if Q_0 is positive semidefinite. A generalization of the QP is the quadratically constrained quadratic program (QCQP):

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2} x^\top Q_0 x + a_0^\top x \\ \text{subject to} \quad & \tfrac{1}{2} x^\top Q_i x + a_i^\top x \leq b_i, \quad i = 1, \ldots, m. \end{aligned} \tag{2.5}$$

A QCQP is convex if and only if all Q_i $(i = 0, \ldots, m)$ are positive semidefinite (Lobo, Vandenberghe, Boyd, and Lebret 1998). Whereas convex QP or even QCQP are commonly solved by reformulations (transformations) to second-order cone programming (SOCP) or semidefinite programming (SDP) (see Section 2.2.3), the ques-

tion how to obtain a reliable global solution for non-convex QCQP is still an active field of research. Details on the necessary transformations to cast convex QCQP into an SOCP or SDP can be found in, e.g., Lobo *et al.* (1998); Alizadeh and Goldfarb (2003); Bao, Sahinidis, and Tawarmalani (2011).

2.2.3 Conic programming

Conic programming refers to a class of problems designed to model convex OPs. The most prominent members of this class are LP, SOCP and SDP. We follow the common practice to define a conic program (CP) as:

$$\begin{aligned} \text{minimize} \quad & a_0^\top x \\ \text{subject to} \quad & Ax + s = b \\ & s \in \mathcal{K}, \end{aligned} \tag{2.6}$$

where the set $\mathcal{K}$ is a nonempty closed convex cone.

The standard form of CP as given in Equation 2.6 minimizes a linear objective over a convex cone ($b - Ax = s \in \mathcal{K}$). As Nemirovski (2006) points out, representing CPs in this form has two main advantages. First, this formulation has strong unifying abilities which means only a few cones allow modeling of many different types of OPs. Additionally, the nonlinearities are no longer represented by general nonlinear objective and constraint functions but vectors and matrices which allows the algorithms to utilize the structure present in the convex OPs. Second, the convexity is built-in into the definition of CPs. At the same time, theoretically, any convex OP can be reformulated into the form given in Equation 2.6. Thereby nonlinear objective functions are expressed in *epigraph form* (see, e.g., Boyd and Vandenberghe 2004):

$$\begin{aligned} \text{minimize} \quad & t \\ \text{subject to} \quad & f_0(x) \leq t \\ & f_i(x) \leq b_i. \end{aligned} \tag{2.7}$$

Practically the number of OPs which can be solved via CP is limited by the number of cones supported by a given optimization solver. State of the art solvers distinguish between up to eight different types of cones. Following the definitions in Diamond and Boyd (2015) and O'Donoghue *et al.* (2016), a convex cone $\mathcal{K}$ is typically a Cartesian product from simple convex cones of the following types.

Zero cone and free cone

The zero and free cones are, respectively, given by

$$\mathcal{K}_{\text{zero}} = \{0\}, \mathcal{K}_{\text{free}} = \mathbb{R} = \mathcal{K}_{\text{zero}}^*,$$

where for a cone $\mathcal{K}$ we write $\mathcal{K}^* = \{y | x^\top y \geq 0 \text{ for all } x \in \mathcal{K}\}$ for the dual cone (see, e.g., Boyd and Vandenberghe (2004) for more information about dual cones). From Equation 2.6 it can be immediately seen, that in the case of linear equality constraints s_i has to be zero, i.e., $s_i \in \mathcal{K}_{\text{zero}} \iff s_i = b_i - a_i^\top x = 0 \iff a_i^\top x = b_i$.

Linear cone (non-negative orthant)

The linear cone is, given by

$$\mathcal{K}_{\text{lin}} = \{x \in \mathbb{R} \mid x \geq 0\}. \tag{2.8}$$

This cone is used to represent linear inequality (less than or equal) constraints, by requiring s_i to be non-negative, i.e., $s_i \in \mathcal{K}_{\text{lin}} \iff s_i = b_i - a_i^\top x \geq 0 \iff a_i^\top x \leq b_i$.

From the definition of the free cone and non-negative cone, it is apparent that any LP can be written as a CP where $\mathcal{K}$ is a product of free and non-negative cones.

Second-order cone

The second-order cone is given by

$$\mathcal{K}_{\text{soc}}^n = \{(t, x) \in \mathbb{R}^n \mid x \in \mathbb{R}^{n-1},\ t \in \mathbb{R},\ ||x||_2 \leq t\}. \tag{2.9}$$

This cone is used to model sums of norms as well as convex QP and QCQP (Lobo *et al.* 1998; Alizadeh and Goldfarb 2003). CPs where $\mathcal{K}$ is a product of free, non-negative and second-order cones are commonly referred to as SOCP.

Positive semidefinite cone

The positive semidefinite (PSD) cone is given by

$$\mathcal{K}_{\text{psd}}^n = \{X \mid X \in \mathcal{S}^n,\ z^\top X z \geq 0 \text{ for all } z \in \mathbb{R}^n\}. \tag{2.10}$$

Here $\mathcal{S}^n$ refers to the space of real-symmetric $n \times n$ matrices. CPs restricted to the positive semidefinite cone are referred to as SDPs. They are commonly used for solving combinatorial problems (e.g., maximum cut problem) and for solving convex QPs and QCQPs (Vandenberghe and Boyd 1996; Helmberg 2000; Freund 2009; Bao *et al.* 2011). Lobo *et al.* (1998) show that each SOCP can be rewritten as an SDP.

Exponential cone

The primal exponential cone is defined as

$$\mathcal{K}_{\text{expp}} = \{(x, y, z) \in \mathbb{R}^3 \mid y > 0,\ ye^{\frac{x}{y}} \leq z\} \cup \{(x, 0, z) \in \mathbb{R}^3 \mid x \leq 0,\ z \geq 0\}. \quad (2.11)$$

Its dual is given by

$$\mathcal{K}_{\text{expd}} = \{(u, v, w) \in \mathbb{R}^3 \mid u < 0, -ue^{\frac{v}{u}} \leq ew\} \cup \{(0, v, w) \in \mathbb{R}^3 \mid v,\ w \geq 0\}. \quad (2.12)$$

As can be inferred from Equation 2.11, the exponential cone can be used to model exponential functions and logarithms. More details about the exponential cone and functions representable by the exponential cone can be found in Chares (2009) and Serrano (2015).

Power cone

The 3-dimensional primal power cone has already been investigated in Koecher (1957) and is defined as

$$\mathcal{K}^{\alpha}_{\text{powp}} = \{(x, y, z) \in \mathbb{R}^3 \mid x,\ y \geq 0,\ x^{\alpha}y^{1-\alpha} \geq |z|\},\ \text{where } \alpha \in [0, 1]. \quad (2.13)$$

Its dual is given by

$$\mathcal{K}^{\alpha}_{\text{powd}} = \left\{(u, v, w) \in \mathbb{R}^3 \mid u,\ v \geq 0,\ \left(\frac{u}{\alpha}\right)^{\alpha}\left(\frac{v}{1-a}\right)^{1-\alpha} \geq |w|\right\},\ \text{where } \alpha \in [0, 1]. \quad (2.14)$$

The power cone can be used to model powers and p-norms. For more information about the power cone and its modeling capabilities we refer to Chares (2009).

Putting the hierarchies described above all together we get the following ordering among OPs

$$\text{LP} \subset \text{convex QP} \subset \text{convex QCQP} \subset \text{SOCP} \subset \text{SDP} \subset \text{CP}.$$

2.2.4 Nonlinear optimization

The most general problem class is nonlinear optimization or nonlinear programming (NLP). This is the problem where at least one f_i, $i = 0, \ldots, m$ in Equation 2.2 is not linear. NLPs are not required to be convex, which makes it in general hard to obtain a reliable global solution. Contrary to the convex case, in a non-convex setting most optimization algorithms only find the extrema of f_0 in the neighborhood of the starting value (local optimum).

2.2.5 Mixed integer programming

A mixed integer program (MIP) adds the additional requirement to the optimization problem that some of the objective variables can only take integer values. Considering Equation 2.2, a problem is called a mixed integer problem if the (type) constraint $x_k \in \mathbb{Z}$ for at least one k is added. In the case where all n objective variables are integral we speak of a pure integer programming (IP) problem. An IP where all variables are bounded between zero and one, i.e., $x \in \{0,1\}^n$, is called a binary (integer) program.

Since MIPs are non-convex, even mixed integer linear programs (MILP) can already be hard to solve. Nevertheless an increase in quantity and quality of free and nonfree solvers was observed in the last decade (Linderoth and Ralphs 2005; Bixby 2012). Typically solvers use branch-and-bound (Land and Doig 1960) and the cutting plane (Gomory 1960) algorithms or a combination of both. The algorithms avoid solving the problem directly, but instead solve multiple relaxations where the integer constraint is dropped.

2.3 Software

Recently, an increase of the available packages handling many different OPs in R has been observed. The CRAN optimization and mathematical programming task view (Theußl *et al.* 2019a) currently lists around 100 different optimization related packages. The capability these packages provide range from solvers which can solve a wide range of optimization problems (e.g., **optimx** (Nash and Varadhan 2011; Nash 2014a)) to very specialized solvers which are created to solve a specific problem type very fast (e.g., nonlinear regression solvers). This section provides an overview of the solver landscape in R. The insights gained in this section will be used to derive a consistent solver infrastructure. First, we investigate the available (open source) solvers, splitting these into linear solvers, quadratic solvers, conic solvers and general purpose solvers. We then discuss commercial solvers (i.e., any solver developed for sale) and the NEOS server.

2.3.1 Overview

As pointed out in Section 2.2, in the field of optimization we are typically facing different problem classes. The possibly three most important distinctions are between linear versus nonlinear problems, integer versus continuous and convex versus non-convex problems.

Ordered based on increasing complexity, an objective function might be of type linear, (convex) quadratic, conic (i.e., any objective expressible as a CP) or func-

tional (i.e., any objective expressible as function). Similarly constraints are typically of type box, linear, (convex) quadratic, conic or functional. Box constraints (or variable bounds) are a special type of linear constraints which enforce lower and upper bounds on the objective variables.

The terms conic objective/constraints are used in a general way and refer to any linear and nonlinear objective/constraints that can be reformulated as a conic problem. Therefore this also includes problems with linear and convex quadratic objective/constraints. Note that, solvers that take as input values a linear objective and conic constraints are also applicable to OPs with conic objective and conic constraints by making use of the epigraph form transformation. The most general form are functional objective/constraints which includes all linear and nonlinear objective/constraints.

Table 2.1 gives an overview on optimization packages available at CRAN (https://CRAN.R-project.org) with a focus on open source solvers. The position of a particular package in the table indicates its ability to solve a given problem. Each problem class to the left and above of the current position can be handled by the package including its current position. For instance, the **ECOSolveR** (Fu and Narasimhan 2019) package which provides an interface to the **ECOS** (Domahidi, Chu, and Boyd 2013) library can solve conic problems restricted to combinations of the zero, non-negative, second-oder and primal exponential cone. Since **ECOS** is equipped with a branch-and-bound algorithm, it can also be used to solve mixed integer conic problems.

2.3.2 The R solver landscape

The solver landscape can be split into two parts. First, solvers where the functional form is fixed and only the coefficients are provided, which includes all LP, QP, QCQP and CP solvers currently available in R. Second, solvers which can optimize any functional form expressible as an R function. This includes most NLP solvers, sometimes summarized as general purpose solvers.

Linear solvers

Interfaces to several open source LP and MILP solvers are available in R. Most of these packages provide a high-level access to the solver, those explicitly designed to provide a low-level access are commonly marked with the suffix **API**.

The Computational Infrastructure for Operations Research (COIN-OR) project (https://www.coin-or.org/) provides an open source software framework for the operations research community including the COIN-OR linear programming (**Clp**, Forrest, de la Nuez, and Lougee-Heimer 2004) and the **SYMPHONY** (Ralphs and

Constraints	Objective			
	linear	quadratic	conic	functional
no				**BB**, **mize**, **trustOptim**
box				**DEoptim**, **dfoptim**, **GenSA**, **lbfgsb3**, **metaheuristicOpt**, **minqa**, **optimx**, **Rcgmin**, **rgenoud**, **Rmalschains**, **Rvmmin**, **soma**, **stats**, **ucminf**
linear	**clpAPI***, **Rglpk**$^{*+}$, **lpSolve**$^{*+}$, **rcdd***, **Rsymphony**$^{*+}$	**coneproj***, **Dykstra***, **kernlab**, **LowRankQP***, **osqp***, **quadprog***, **ROI.plugin.qpoases**		
quadratic				
conic			**cccp***, **CLSOCP***, **ECOSolveR**$^{*+}$, **Rcsdp***, **Rdsdp***, **scs***	
functional				**alabama**, **deoptimr**, **clue**, **NlcOptim**, **nloptr**, **Rsolnp**

Table 2.1: Selected R packages displayed based on the types of optimization problems they are applicable to. Here * indicates that the solver is restricted to convex problems and $^{+}$ indicates that the solver can model integer constraints.

Güzelsoy 2005, 2011) solver. **Clp** is mainly used as library and provides methods for solving LPs via interior point methods or the simplex algorithm. In R **Clp** is available through **clpAPI** (Roettger, Gelius-Dietrich, and Fritzemeier 2019) which provides a low level interface to **Clp**. **SYMPHONY** is a flexible MILP solver written in C++, that transforms the MILP into LP relaxations to be solved by any LP solver callable through the Open Solver Interface (OSI). **Rsymphony** (Hornik, Harter, and Theußl 2017a) provides an interface to the **SYMPHONY** solver, where by default

the LP relaxations are solved by the **Clp** solver.

GNU Linear Programming Kit (**GLPK**, Makhorin 2011) is a solver library written in ANSI C, for solving LP and MILP. The low level interface **glpkAPI** (Roettger, Gelius-Dietrich, and Luangkesorn 2018) and the high level interface **Rglpk** (Theußl and Hornik 2019) are available in R.

lp_solve (Berkelaar, Eikland, and Notebaert 2016) uses the simplex algorithm combined with branch-and-bound to solve LPs and MILPs. It furthermore allows modeling of semi-continuous and special ordered sets problems. Packages **lpSolve** (Berkelaar 2019) and **lpSolveAPI** (Konis 2019) provide access to the **lp_solve** solver in R.

Additionally the function `lpcdd()` from package **rcdd** (Geyer and Meeden 2019) and the function `simplex()` from package **boot** (Canty and Ripley 2019) can be used to solve LPs via the simplex algorithm.

By taking a closer look at the elements needed by packages capable of solving LPs and MILPs[1] we can conclude that the following elements should be present in a consistent and convenient optimization infrastructure for modeling LPs and MILPs.

`objective`: A numeric vector giving the coefficients of the linear objective.

`constraints`:

- Includes a constraint matrix A (see Equation 2.3),
- a vector giving the direction of the constraints (i.e., `==`, `<=` or `>=`), and
- a vector giving the right hand side b (see Equation 2.3).

`bounds`: Two vectors giving the lower and upper bounds.

`types`: A vector storing the type information, i.e., binary, integer and numeric.

`maximum`: A boolean indicating if the objective function should be maximized or minimized.

Note that the elements `bounds` and `maximum`, as well as the constraint directions and the binary types are not strictly necessary. Their inclusion is motivated by the fact that they are supported by many solvers and simplify the problem specification.

Quadratic solvers

As Table 2.1 shows, most of the quadratic solvers are designed to solve convex quadratic problems with linear constraints. The **quadprog** (Turlach and Weingessel 2019) package uses the dual method described in Goldfarb and Idnani (1983).

[1] This includes commercial and non-commercial solvers.

LowRankQP (Ormerod and Wand 2018) is based on an interior point algorithm described in Fine and Scheinberg (2001). **Dykstra** (Helwig 2018) implements Dykstra's cyclic projection algorithm (Dykstra 1983), **coneproj** (Meyer and Liao 2018) transforms the original QP problem into a cone projection problem (Liao and Meyer 2014) and **osqp** (Stellato, Banjac, Goulart, and Boyd 2019) uses the alternating direction method of multipliers described in Stellato, Banjac, Goulart, Bemporad, and Boyd (2017).

Additionally, the package **ROI.plugin.qpoases** (Schwendinger 2018) and the function `ipop()` from **kernlab** (Karatzoglou, Smola, Hornik, and Zeileis 2004; Karatzoglou, Smola, and Hornik 2019) can be used to obtain solutions for non-convex quadratic problems with linear constraints. However, for the non-convex case there is no guarantee that the returned solution is a global optimum. **ROI.plugin.qpoases** is an interface to the **qpOASES** (Ferreau, Kirches, Potschka, Bock, and Diehl 2014; Ferreau, Potschka, and Kirches 2017) library, which uses an online active set strategy to solve quadratic optimization problems.

QP solvers generally take the same arguments as LP solvers plus an additional matrix parameter storing the coefficients of the quadratic term Q_0.

Conic solvers

Most of the conic solvers use a standard form similar to Equation 2.6, where the objective function is assumed to be linear and the vector $b - Ax$ is restricted to a certain cone $\mathcal{K}$. Nevertheless, in Table 2.1 they are shown to have a conic objective function and conic constraints to express that they are able to solve any LP and convex NLP expressible by a CP. Therefore, which types of NLPs a given solver can solve, depends on the types of cones the solver can model. Table 2.2 shows the conic solvers available in R and the types of cones they support.

	zero	linear	second order	semidefinite	exponential	dual exponential	power	dual power
CLSOCP	✓	✓	✓					
cccp	✓	✓	✓	✓				
ECOSolveR	✓	✓	✓		✓			
Rcsdp	✓	✓	✓	✓				
Rdsdp	✓	✓	✓	✓				
scs	✓	✓	✓	✓	✓	✓	✓	✓

Table 2.2: Conic packages and the supported cones.

Package **CLSOCP** (Rudy 2011) is specialized in solving SOCPs, it is a pure R

implementation of the one-step smoothing Newton method based on the algorithm described in Tang, He, Dong, and Fang (2012). For solving SDP there exist the specialized packages **Rcsdp** and **Rdsdp**. Since any SOCP can be transformed into an SDP they can also be used for solving SOCPs. **Rcsdp** (Bravo 2016) is an interface to the **CSDP** (Borchers 1999) library which is part of the COIN-OR project. **Rdsdp** (Zhu and Ye 2016) is an interface to the **DSDP** (Benson and Ye 2008) library. Both packages can read and **Rcsdp** can also write `sdpa` files, which is a file format commonly used to store SDPs. The **cccp** (Pfaff 2015) package provides functions to solve LPs, QPs, SOCPs and SDPs, the algorithms are reported to be similar to those in **CVXOPT** (Andersen *et al.* 2019). **CVXOPT** is a Python package for solving convex OPs via interior-point methods (more information about the algorithms can be found in Andersen, Dahl, Liu, and Vandenberghe 2012). **ECOSolveR** (Fu and Narasimhan 2019) is an interface to the embedded conic solver **ECOS** (Domahidi *et al.* 2013). A special feature of **ECOS** is that it combines convex optimization with branch-and-bound techniques, therefore it can be used to solve CPs where some variables are required to be integer. The **scs** (Schwendinger and O'Donoghue 2019) package is an interface to the Splitting Conic Solver (**SCS**, O'Donoghue 2015) library, which uses a version of the alternating direction method of multipliers (ADMM) for solving CPs. **SCS** is designed to solve large cone problems faster than standard interior-point methods. More information about the algorithm and a comparison to other solvers can be found in O'Donoghue *et al.* (2016).

General purpose solvers

Solvers capable of handling nonlinear objective functions without further restrictions are called general purpose solvers (GPS). These solvers can minimize (or maximize) any functional form representable as an R function with — depending on solver capabilities — different types of constraints, where again the most general form of constraint is the functional constraint (i.e., an R function). The generality of GPS comes at the price of performance and that there is usually no guarantee that a global optimum is reached.

	Global GPS		Local GPS	
	Gradient free	Gradient	Gradient free	Gradient
No Constraint	6	0	7	19
Box Constraint	28	4	8	12
Functional Constraint	2	0	7	7

Table 2.3: Overview of general purpose solvers.

Important properties of GPS are whether they are designed to search for a local or global optimum, if gradient information has to be provided or the method is

gradient free and which type of constraints can be set. Table 2.3 shows the number of GPS methods grouped by these properties (the counts are based on Table A.2 where additional details can be found) and reveals some interesting details about the R GPS landscape. Around 60 percent of the GPS are designed for local optimization, even though most of the local solvers utilize gradient information, only four of the global solvers use gradient information. The difference in distribution of gradient based and gradient free optimization algorithms between global and local GPS can be explained by the fact that in global optimization, metaheuristics like evolutionary methods or particle swarm optimization are commonly used. In a recent study Mullen (2014a) surveys the continuous global optimization packages available in R and compares their performance on a set of tests bundled in the **globalOptTests** (Mullen 2014b) package. Table A.2 gives an extensive listing of which methods are designed to search for a global solution. For more information about the methods we refer to Mullen (2014a).

Based on the type of constraints, the GPS can be divided into *no constraints*, *box constraints*, *linear constraints*, *quadratic constraints* and *functional constraints*. As Table 2.3 shows, most of the GPS support no constraint or box constraints. Fortunately, package **optimx** (Nash and Varadhan 2011) provides a unified interface to many of these solvers, consolidating methods from packages **stats**, **ucminf** (Nielsen and Mortensen 2016), **minqa** (Bates, Mullen, Nash, and Varadhan 2014), **Rcgmin** (Nash 2014b), **Rvmmin** (Nash 2018) and **BB** (Varadhan and Gilbert 2009). It was designed as a possible successor of `optim` which is part of the **stats** package and can be used to solve OPs with box constraints. Another package which incorporates many different algorithms is **nloptr** (Ypma, Borchers, and Eddelbuettel 2018). It is an R interface to the **NLopt** (Johnson 2019) library, which bundles several global and local optimization algorithms. Depending on the algorithm it can solve NLPs with box-constraints or functional-constraints.

Most of the GPS able to handle functional constraints allow to specify functional equality and/or functional inequality constraints. Since specifications may vary from one solver to the other a general optimization infrastructure should be designed in a way that the functional form employed can be transformed into the commonly used forms.

To model functional equality constraints the following two forms are most commonly used:

- $h_i(x) = 0,\ i = 1, \ldots, k$ (e.g., **alabama** (Varadhan 2015), **DEoptimR** (Conceicao 2016), **NlcOptim** (Chen and Yin 2019), **nloptr::auglag**, **nloptr::isres**, **nloptr::slsqp**, and **Rnlminb2** (Würtz 2014)[2])
- $h_i(x) = b_i,\ i = 1, \ldots, k$ (e.g., **Rsolnp** (Ghalanos and Theußl 2015))

where h is a function and $b \in \mathbb{R}^k$ gives the right hand side. Similarly, functional inequality constraints are commonly given in one of the following three forms:

- $g_j(x) \leq 0,\ j = k+1, \ldots, m$ (e.g., **DEoptimR**, **nloptr::nloptr**, **NlcOptim**, **Rnlminb2**, **csr::snomadr** (Racine and Nie 2018))
- $g_j(x) \geq 0,\ j = k+1, \ldots, m$ (e.g., **alabama**, **neldermead** (Bihorel and Baudin 2018), **nloptr::auglag**, **nloptr::cobyla**, **nloptr::ires**, **nloptr::mma**, and **nloptr::slsqp**)
- $l_j \leq g_j(x) \leq u_j,\ j = k+1, \ldots, m$ (e.g., **ipoptr** (Ypma 2011)[2], **Rdonlp2** (Würtz 2017)[2], **Rsolnp**).

where g is a function and $l \in \mathbb{R}^{m-k}$, $u \in \mathbb{R}^{m-k}$ are the lower and upper bounds of the constraints.

An analysis of the above solver spectrum reveals that the critical arguments to GPS are:

`start`: The initial values for the (numeric) parameter vector.

`objective`: The function to be optimized.

`constraints`: Depending on the GPS the constraints can be none, linear, quadratic, functional equality or functional inequality constraints. To model functional constraints consistently with linear and quadratic constraints the following elements are needed.

- A function representing the constraints,
- a vector giving the direction of the constraints, and
- a vector giving the right hand side.

`bounds`: Variable bounds, commonly given as lower and upper bounds.

Additionally some GPS make use of the gradient and/or Hessian of the objective and the Jacobian of the constraints. The optional elements can be summarized by:

[2]Note the packages **ipoptr**, **Rnlminb2**, and **Rdonlp2** are not available on CRAN but on R-Forge. Information about the installation of **ipoptr** can be found at https://www.coin-or.org/Ipopt/documentation/node10.html.

`gradient`: A function that evaluates the gradient of the argument `objective`.

`hessian`: A function that evaluates the Hessian of the argument `objective`.

`jacobian`: A function that evaluates the Jacobian of the argument `constraints`.

`maximum`: A boolean indicating whether the objective function should be maximized or minimized.

`control`: Further control arguments specific to the solver.

Return values include:

`par`: The "solution" (parameters) found.

`value/objective`: The value of the objective function evaluated at the "solution".

`convergence, status`: An integer information about the convergence and exit status of the optimization task.

`gradient`: The gradient evaluated at the solution found.

`hessian`: The Hessian evaluated at the solution found.

`message`: A text message giving additional information about the optimization / exit status.

`iterations/evaluations`: The number of iterations and / or evaluations.

2.3.3 Other optimization back-ends

Commercial solvers

Since commercial solver packages often bundle a variety of solvers, it is often not possible to assign them to a certain problem class. At the time of this writing R interfaces are available to the commercial solver software **CPLEX** (ILOG 2019), **MOSEK** (ApS 2019), **Gurobi** (Gurobi Optimization, LLC 2018), **Lindo** (Lindo Systems 2013) and **localsolver** (Benoist, Estellon, Gardi, Megel, and Nouioua 2011). To cover also the commercial side, the interface packages **Rcplex** (Theußl and Bravo 2016) and **Rmosek** (Friberg 2019) were included in defining the requirements for a consistent solver infrastructure.

Network-Enabled Optimization System (NEOS)

The NEOS (Czyzyk, Mesnier, and Moré 1998; Dolan 2001; Gropp and Moré 1997) server (`https://neos-server.org`) provides free access to more than 60 numerical optimization solvers. It can be accessed via the internet by submitting OPs via the homepage, email, the XML-RPC application programming interface or Kestrel. Depending on the solver the OPs have to be formulated in different ways, overall most solver support input from AMPL and GAMS. The R packages **rneos** (Pfaff 2017) and **ROI.plugin.neos** (Hochreiter and Schwendinger 2019) use the XML-RPC API to communicate with the NEOS server. In order to upload OPs with **rneos**, the OPs have to be formulated in the input format supported by the solver (e.g., AMPL, GAMS, MPS). In contrast to that, **ROI.plugin.neos** supports OPs generated with **ROI**, internally each OP is transformed to GAMS before they are submitted to the server. The result is again converted back into a suitable solution format.

2.4 A general optimization infrastructure for R

After reviewing the optimization resources available in R, it is apparent that the main function of a general optimization infrastructure package should take at least three arguments:

problem representing an object containing the description of the corresponding OP,

solver specifying the solver to be used (e.g., `"glpk"`, `"nlminb"`, `"scs"`),

control containing a list of additional control arguments to the corresponding solver.

The arguments `solver` and `control` are easily understood, since from the available solver spectrum we only have to choose those which are capable to handle the corresponding OP and (optionally) supply appropriate control parameters. However, building the `problem` object, in a general and intuitive way, is a challenging task which leads to several design issues.

Based on the review in Sections 2.2 and 2.3 it seems natural to instantiate OPs based on an *objective* function, one or several *constraints*, *types* and *bounds* of the objective variables, as well as the direction of optimization (whether a minimum or a *maximum* is sought).

In the remainder of this section we discuss the conceptual design of **ROI** and how to use it to formulate optimization problems. For illustrative purposes we already use functionality of package **ROI**.

```
R> library("ROI")
```

2.4.1 Objective function

The survey of optimization solvers in Section 2.3 reveals that the way the objective function is stored depends primarily on its functional form. If the objective function is linear (L), i.e., $a_0^\top x$, then it is common practice to only supply a coefficient vector $a_0 \in \mathbb{R}^n$. For quadratic objective functions (Q) of the form $\frac{1}{2}x^\top Q_0 x + a_0^\top x$ most solvers take a vector $a_0 \in \mathbb{R}^n$ and a matrix $Q_0 \in \mathbb{R}^{n \times n}$ as input. General nonlinear objective functions (F, i.e., nonlinear functions which cannot be represented as an QP or CP), are represented as an R function which takes the vector of objective variables as argument and returns the objective value. Depending on the type of the objective function, i.e., `F`, `Q`, or `L`, only a subset of the solver spectrum can be used.

Objective function types and corresponding constructors implemented in **ROI** are:

F The most general form of an objective function is created with the constructor `F_objective(F, n, G, H, names)` by simply supplying `F`, an R function representing $f_0(x)$, and n the length of x. Optionally, information about the gradient and the Hessian can be provided via the arguments `G` and `H`. If no gradient is provided it will be calculated numerically if needed. The optional `names` argument is propagated to the solution object to make the solution more readable.

Q Objective functions representing a quadratic form as outlined above can be easily created with the `Q_objective(Q, L, names)` constructor taking `Q`, the quadratic part Q_0, and optionally `L`, the linear part a_0, as arguments. The `names` argument is again optional.

L If the objective to be optimized is a linear function then one should use the `L_objective(L, names)` constructor supplying `L` (the coefficients of the objective variables) as a numeric vector. The `names` argument is again optional.

All three constructors return an object inheriting from class '`objective`'.

2.4.2 Constraints

To model all the problem classes introduced in Section 2.2, four different types of constraints are sufficient. Thereby arguments with the same name have the same functionality irrespective of the constraint type and will therefore be explained only once.

F The most general form of constraints can express any constraint representable by an R function. They are created via `F_constraint(F, dir, rhs, J,`

`names)`. Here `F` is either a function or a list of functions, `dir` is a character vector giving the direction of the constraint and `rhs` is a numeric vector giving the right hand side of the constraint. The optional arguments `J` and `names` can be used to provide the Jacobian and the variable names of the constraints.

C Conic constraints are constructed via the function `C_constraint(L, cones, rhs, names)`, where `L` can be either a numeric vector of length n or a matrix of dimension $m \times n$. In accordance with Equation 2.6 the `cones` impose a restriction on the slack variable s. A conic constraint can be comprised of several cones, where each cone type can occur multiple times. The cone constructors all start with `K_` followed by a short cut of the cone name, as defined in Section 2.2.3. Currently **ROI** implements constructors for the cones `K_zero`, `K_lin`, `K_soc`, `K_psd`, `K_expp`, `K_expd`, `K_powp`, and `K_powd`. To combine different cones the generic combine `c()` can be used.

Q Quadratic constraints as defined in Equation 2.5 can be easily created with the constructor `Q_constraint(Q, L, dir, rhs, names)`. The quadratic constraints `Q` are given as a list of length m where the entries are either of $n \times n$ matrices or `NULL`.

L Linear constraints are constructed via the function `L_constraint(L, dir, rhs, names)`.

All constructors return an object inheriting from class '`constraint`'. Since in many situations it is desirable to optimize a given objective function subject to composite constraints of different kinds, **ROI** can combine multiple constraints into a single constraint using the generic functions `c()` or `rbind()`. Since the matrices `L` and `Q` can become huge but are typically sparse we use the simple triplet matrix format from the **slam** (Hornik, Meyer, and Buchta 2019) package to store them internally. In the simple triplet matrix format (sometimes referred to as coordinate list format) only the row indices, the column indices and the values of the non-zero elements are stored in a list. We choose the **slam** package not only for efficiency reasons but also due to the fact that many solver APIs demand such a format.

2.4.3 Objective variable types

As it is common practice in mixed-integer solvers to distinguish between the variable types continuous, integer and binary we encode the variable choice with the following characters: `"C"` for continuous, `"I"` for integer and `"B"` for binary. By default all the variables are assumed to be of continuous type.

2.4.4 Bounds

A variable bound is a special type of constraint used to restrict an objective variable between a real lower and upper bound. Therefore, variable bounds are often also called "box bounds" or "box constraints". Although variable bounds could be easily modeled as linear constraints, many GPS only support variable bounds (see Table 2.3). Furthermore, most solvers that support any type of constraint allow to specify variable bounds directly. Thus, it is reasonable but also convenient to consider them separately.

Typically, implementations of optimization algorithms differentiate between five types of objective variable bounds: free $(-\infty, \infty)$, upper $(-\infty, ub]$, lower $[lb, \infty)$, double bounded $[lb, ub]$, and fixed bounds. In **ROI** variable bounds are represented as a list with two elements—`upper` and `lower`, where only the non-default values are stored in a simple sparse format. In this sparse format only indices and the non-default values are stored. For the lower bounds the default value is zero and for the upper bounds the default value is infinity. Thus, for OPs where all the variables are required to take values in the interval $[0, \infty)$ no bounds have to be specified. Upper and/or lower bounds are specified by providing the index i of the corresponding variable (arguments `li`, `ui`) and its lower (`lb`) or upper (`ub`) bound, respectively. Therefore, the box constraints $-\infty \leq x_1 \leq 4$, $0 \leq x_2 \leq 100$, $2 \leq x_3 \leq \infty$ and $0 \leq x_4 \leq \infty$ are constructed in **ROI** as follows:

```
R> V_bound(li = 1:4,  ui = 1:4, lb = c(-Inf, 0, 2, 0),
+    ub = c(4, 100, Inf, Inf))

ROI Variable Bounds:


2 lower and 2 upper non-standard variable bounds.
```

If all upper and lower values are provided (default values are not omitted) the indices can be left out:

```
R> V_bound(lb = c(-Inf, 0, 2, 0), ub = c(4, 100, Inf, Inf))

ROI Variable Bounds:


2 lower and 2 upper non-standard variable bounds.
```

If the default values are omitted the number of objective variables has to be provided.

```
R> V_bound(li = c(1L, 3L),  ui = c(1L, 2L), lb = c(-Inf, 2),
+          ub = c(4, 100), nobj = 4L)
```

```
ROI Variable Bounds:

2 lower and 2 upper non-standard variable bounds.
```

Consider the box constrains $0 \leq x_3 \leq 20$ and $-20 \leq x_i \leq 20$ for all $i \in I = \{1, 2, 4\}$. The default lower and upper bound can be changed by the arguments `ld` and `ud`. Thus, this variable bound can be constructed by the following **ROI** code.

```
R> V_bound(li = 3, lb = 0, ld = -20, ud = 20, nobj = 4L)

ROI Variable Bounds:

3 lower and 4 upper non-standard variable bounds.
```

2.4.5 Optimization problem

In **ROI**, a new optimization problem is created by calling

```
OP(objective, constraints, types, bounds, maximum).
```

As an example consider the LP

$$\begin{array}{rrcrcrcl} \underset{x}{\text{maximize}} & 3x_1 & + & 7x_2 & - & 12x_3 & & \\ \text{subject to} & 5x_1 & + & 7x_2 & + & 2x_3 & \leq & 61 \\ & 3x_1 & + & 2x_2 & - & 9x_3 & \leq & 35 \\ & x_1 & + & 3x_2 & + & x_3 & \leq & 31 \\ & x_1, x_2 \geq 0, \; x_3 \in [-10, 10]. & & & & & & \end{array} \tag{2.15}$$

This problem can be constructed by the following **ROI** code.

```
R> A <- rbind(c(5, 7,  2), c(3, 2, -9), c(1, 3,  1))
R> dir <- c("<=", "<=", "<=")
R> rhs <- c(61, 35, 31)
R> lp <- OP(objective = L_objective(c(3, 7, -12)),
+    constraints = L_constraint(A, dir = dir, rhs = rhs),
+    bounds = V_bound(li = 3, ui = 3, lb = -10, ub = 10, nobj = 3),
+    maximum = TRUE)
```

Alternatively, an OP can be formulated piece by piece, by creating an empty OP

```
R> lp <- OP()
```

and using the setter/getter functions `objective()`, `constraints()`, `bounds`(), `types()`, and `maximum()` to set/get the corresponding element.

```
R> objective(lp) <- L_objective(c(3, 7, -12))
R> constraints(lp) <- L_constraint(A, dir = c("<=", "<=", "<="),
+                                   rhs = rhs)
R> bounds(lp) <- V_bound(li = 3, ui = 3, lb = -10, ub = 10,
+                        nobj = 3)
R> maximum(lp) <- TRUE
R> lp

ROI Optimization Problem:

Maximize a linear objective function of length 3 with
- 3 continuous objective variables,

subject to
- 3 constraints of type linear.
- 1 lower and 1 upper non-standard variable bound.
```

The setter functions make it easy to alter previously created OPs. The getter function `objective()` returns the objective as function, which can be directly used to evaluate parameters. The number of parameters required, can be obtained by the generic function `length()`.

```
R> param <- rep.int(1, length(objective(lp)))
R> objective(lp)(param)

[1] -2
```

To access the data of the objective, the generic function `terms()` should be used.

```
R> terms(objective(lp))

$L
A 1x3 simple triplet matrix.

$names
NULL
```

For all the other elements the corresponding getter function returns directly the underlying data representation.

Function `OP()` always returns an S3 object of class '`OP`' which stores the entire OP. Storing the OP in a single R object has many advantages, among others:

- the OP can be checked for consistency during the creation of the problem,

- the different elements of the OP can be easily accessed after the creation of the problem,
- and the OP can be easily altered, e.g., a constraint can be added, bounds can be changed, without the need to redefining the entire OP.

The consistency checks verify that the dimensions of the arguments fit together.

2.5 Package ROI

The R optimization infrastructure is structured into the package **ROI** and its accompanying extensions (plug-ins and models). Package **ROI** provides all the necessary classes and methods and manages the extensions (i.e., automatically loads plug-ins and manages meta data about the plug-ins). The extension packages add optimization solvers, read/write functions and additional resources (e.g., model collections). The plug-in extensions play a special role, hence all plug-ins are loaded automatically when **ROI** is loaded. When a plug-in is loaded it provides data about its capabilities. This data is stored in an in-memory database and includes information about to which problems the plug-in is applicable, which formats it can read/write and the control arguments available from the solver and how the solver specific control arguments relate to arguments commonly used.

This mechanism makes it possible that **ROI** is aware of all the installed plug-ins, without the need to change **ROI** when a new plug-in is added. To make the automatic loading possible the plug-ins have to follow the name convention `ROI.plugin.`<name>, where <name> is typically the name of an optimization solver (e.g., **ROI.plugin.glpk** (Theußl 2017)). The prefix `ROI.models` is used for data packages with predefined OPs (e.g., **ROI.models.netlib** (Schwendinger 2019)). In Section 2.5.6 we give an overview about the data packages available in the **ROI** format.

2.5.1 Solving optimization problems

After formulating an OP as described in Section 2.4, it can be solved by calling the function `ROI_solve(x, solver, control, ...)`. This function takes an R object of class '`OP`' containing the formulation of the OP, the name of the solver to be used and a list containing solver specific parameters as arguments. The `solver` and `control` arguments are optional, if no `solver` argument is provided **ROI** will choose an applicable solver automatically (see Section 2.5.7). Alternatively the solver specific parameters can be specified via the dots arguments.

The problem stated in Equation 2.15 can be solved by the following **ROI** code:

```
R> (lp_sol <- ROI_solve(lp, solver = "glpk"))
```

```
Optimal solution found.
The objective value is: 8.670149e+01
```

2.5.2 Solution and status code

Status code

Solver status codes are used to inform the user about the exit status of the solver. Despite the common usage of status codes in optimization solvers there is no widely used standard. Nevertheless, we believe it is desirable to provide unified status codes. The status codes used in `ROI_solve` are simple and consistent with the common practice, to return `0` on success (if a "solution" meeting the solver specific requirements was found) `1` otherwise. For optimization solvers specialized on solving convex LPs, QPs and CPs it can be assumed that a global solution was found, if the status code is `0`. Unfortunately, the same is not true for non-convex QPs and GPS, here status codes are less informative. Some packages like **optimx** and **alabama** check the Karush-Kuhn-Tucker (KKT) conditions to verify that the solution found meets the criteria of a local minimum.

Solution

To make the solutions of the various solvers easy to understand, all the solutions are canonicalized within the plug-ins. After the canonicalization each solution contains the following components:

`solution` the solution of the OP,

`objval` the optimal objective value,

`status` the canonicalized status code,

`message` the original solver message,

and a meta attribute containing the solver name and additional optional arguments.

To obtain the (primal) "solution" one can use the generic function `solution(x, type)`:

```
R> solution(lp_sol)
```

```
[1]  0.000000  9.238806 -1.835821
```

Some OPs have multiple solutions, in the case of BLP (MILP) some solvers can retrieve all (multiple) solutions. For MILPs it is in general not possible to obtain all the solutions but only multiple solutions, since even this simple MILP

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & x_1 - x_2 \\ \text{subject to} \quad & x_1 - x_2 = 0 \\ & x_1, x_2 \in \mathbb{Z} \end{aligned} \tag{2.16}$$

has an infinite number of solutions. The following example is based on Fischetti and Salvagnin (2010) and will be used later to illustrate how multiple solutions can be obtained.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & -x_1 - x_2 - x_3 - x_4 - 99x_5 \\ \text{subject to} \quad & x_1 + x_2 \leq 1 \\ & x_3 + x_4 \leq 1 \\ & x_4 + x_5 \leq 1 \\ & x_i \in \{0, 1\} \end{aligned} \tag{2.17}$$

The `"msbinlp"` solver allows to retrieve all the solutions to the OP defined in Equation 2.17, here `method` gives the solver used within the inner loop and `nsol_max` the maximal number of solutions to be returned. Since we have a pure binary problem and five objective variables, we set `nsol_max` to 32,

```
R> blp <- OP(objective = L_objective(c(-1, -1, -1, -1, -99)),
+    constraints = L_constraint(L = rbind(c(1, 1, 0, 0, 0),
+      c(0, 0, 1, 1, 0), c(0, 0, 0, 1, 1)),
+      dir = c("<=", "<=", "<="), rhs = rep.int(1, 3)),
+      types = rep("B", 5L))

R> (blp_sol <- ROI_solve(blp, solver = "msbinlp", method = "glpk",
+    nsol_max = 32))

2 optimal solutions found.
The objective value is: -1.010000e+02

R> solution(blp_sol)

[[1]]
[1] 1 0 1 0 1

[[2]]
[1] 0 1 1 0 1
```

alternatively it is also possible to set `nsol_max` to `Inf`. This is advantageous if an upper bound on the number of solutions is hard to guess and all the solutions should be retrieved .

If the status code is not zero, `solution` will return `NA`, to prevent the user from using solutions with a status code different from `0`.

```
R> lp_inaccurate_sol <- ROI_solve(lp, solver = "scs", tol = 1e-32)
R> solution(lp_inaccurate_sol)

[1] NA NA NA
```

However in a few situations it can be desirable to obtain solutions even if the solver signals no success. In these cases **ROI** can be forced to return the solution provided by the solver regardless of the status code.

```
R> solution(lp_inaccurate_sol, force = TRUE)

[1]  3.214054e-15  9.238806e+00 -1.835821e+00
```

The "solution" to the dual problem can be retrieved by:

```
R> solution(lp_sol, type = "dual")

[1] -4.298507  0.000000  0.000000
```

Furthermore, `solution()` can be employed to retrieve auxiliary variables,

```
R> solution(lp_sol, type = "aux")

$primal
[1] 61.0000 35.0000 25.8806


$dual
[1] 0.5820896 1.4626866 0.0000000
```

the original solver message,

```
R> solution(lp_sol, type = "msg")

$optimum
[1] 86.70149


$solution
[1]  0.000000  9.238806 -1.835821
```

```
$status
[1] 5

$solution_dual
[1] -4.298507  0.000000  0.000000

$auxiliary
$auxiliary$primal
[1] 61.0000 35.0000 25.8806

$auxiliary$dual
[1] 0.5820896 1.4626866 0.0000000



$sensitivity_report
[1] NA
```

the objective value,

```
R> solution(lp_sol, type = "objval")

[1] 86.70149
```

the status,

```
R> solution(lp_sol, type = "status")

$code
[1] 0

$msg
  solver glpk
    code 5
  symbol GLP_OPT
 message Solution is optimal.
roi_code 0
```

and the status code

```
R> solution(lp_sol, type = "status_code")

[1] 0
```

of the OP.

2.5.3 Reformulations

Reformulations are often used to transform a problem of class A into a problem of class B, where the solution of the original problem can be derived from the solution of the reformulation (which is typically easier to solve). Although reformulation techniques are commonly used in optimization, the functions performing these reformulations are generally hidden within the optimization software. To facilitate the comparison of different reformulation algorithms, **ROI** provides functions for managing reformulations. Function `ROI_registered_reformulations()` lists the available reformulations and `ROI_reformulate(x, to, method)` performs the reformulation. Following Boros and Hammer (2002) we illustrate the transformation of a binary QP into a MILP. The code for the reformulation is based on the implementation in the **relations** (Meyer and Hornik 2019) package.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & 6 - x - 4y - z + 3xy + yz \\ & x, y, z \in \{0, 1\} \end{aligned} \tag{2.18}$$

```
R> Q <- rbind(c(0, 3, 0), c(0, 0, 1), c(0, 0, 0))
R> bqp <- OP(Q_objective(Q = Q + t(Q), L = c(-1, -4, -1)),
+            types = rep("B", 3))
R> glpk_signature <- ROI_solver_signature("glpk")
R> head(glpk_signature, 3)
```

```
  objective constraints bounds cones maximum     C     I     B
1         L           X      X     X    TRUE  TRUE FALSE FALSE
2         L           L      X     X    TRUE  TRUE FALSE FALSE
3         L           X      X     X    TRUE FALSE  TRUE FALSE
```

```
R> milp <- ROI_reformulate(x = bqp, to = glpk_signature)
R> ROI_solve(milp, solver = "glpk")
```

```
Optimal solution found.
The objective value is: -4.000000e+00
```

Here **ROI** selects the applicable reformulations based on the provided signatures. A method is considered to be applicable if it can transform the given OP into a new OP, where the signature of the new OP is a subset of the signature provided in the argument `to`. Since it is possible that several methods are applicable, the argument `method` can be used to select a specific reformulation method.

2.5.4 ROI solvers

ROI can currently make use of more than thirty different solvers, applicable to a wide range of OPs. Inspired by R's `available.packages()` function, **ROI** can return a listing of the solver plug-ins available at CRAN, R-Forge (https://r-forge.r-project.org/, Theußl and Zeileis 2009) and GitHub (https://github.com/). `ROI_available_solvers()` without an argument lists all the available solvers. If an OP is provided as argument, only the available solvers applicable will be returned.

```
R> ROI_available_solvers(bqp)[, c("Package", "Repository")]
```

```
             Package                                  Repository
6    ROI.plugin.neos https://r-forge.r-project.org/src/contrib
10  ROI.plugin.cplex              http://R-Forge.R-project.org
14 ROI.plugin.gurobi              http://R-Forge.R-project.org
17  ROI.plugin.mosek              http://R-Forge.R-project.org
19   ROI.plugin.neos              http://R-Forge.R-project.org
28 ROI.plugin.gurobi    https://github.com/FlorianSchwendinger
29  ROI.plugin.mosek    https://github.com/FlorianSchwendinger
```

A listing of all the available plug-ins on CRAN and R-Forge could be easily compiled by just using the `available.packages()` function. But to be able to find all the solvers available and applicable to a given OP also the solver signature is needed. Therefore a database (an rds file on R-Forge) containing the solver signatures and the information provided by `available.packages()` was compiled and is queried whenever `ROI_available_solvers()` is called.

A vector of all solvers installed and loaded (registered) can be obtained by `ROI_registered_solvers()`,

```
R> head(ROI_registered_solvers(), 3)
```

```
             nlminb              alabama                cplex
 "ROI.plugin.nlminb" "ROI.plugin.alabama"   "ROI.plugin.cplex"
```

similarly

```
R> ROI_applicable_solvers(lp)
```

```
[1] "alabama"  "cplex"    "ecos"     "glpk"     "lpsolve"  "neos"
[7] "qpoases"  "scs"      "symphony"
```

returns a vector giving the names of the registered solvers applicable to a given problem. Both return values are based on the solver registry, which stores the solver

Constraints	Objective: linear	Objective: quadratic	Objective: conic	Objective: functional
no				
box				**optimx**
linear	**clp***, **cbc**$^{*+}$, **glpk**$^{*+}$, **lpsolve**$^{*+}$, **msbinlp**$^{*+}$, **symphony**$^{*+}$	**ipop**, **quadprog***, **qpoases**		
quadratic		**cplex**$^{+}$, **gurobi**$^{*+}$, **mosek**$^{*+}$, **neos**$^{+}$		
conic			**ecos**$^{*+}$, **scs***	
functional				**alabama**, **deoptim**, **nlminb**, **nloptr**

Table 2.4: Currently available **ROI** plug-ins displayed based on the types of optimization problems they are applicable to. Here * indicates that the solver is restricted to convex problems and $^{+}$ indicates that the solver can model integer constraints. Note all the plug-ins have the prefix '`ROI.plugin`' and the modeling capabilities of the plug-ins do not necessarily represent the modeling capabilities of the underlying solvers.

method and information about the solver registered by the plug-ins. The solver registry is an in-memory database based on the **registry** (Meyer 2019) package.

`ROI_installed_solvers()` gives a listing of all the installed plug-ins (not necessarily loaded) delivered directly with **ROI** and found by searching for the prefix '`ROI.plugin`' in the R library trees.

```
R> head(ROI_installed_solvers(), 3)
```

```
                nlminb                 alabama                   cplex
"ROI.plugin.nlminb" "ROI.plugin.alabama"    "ROI.plugin.cplex"
```

An overview on the currently available solver plug-ins based on the problem types is given in Table 2.4. Please note that the functionality provided in a plug-in does not necessarily have to be the same as the functionality of the solver, e.g.,

ROI.plugin.nlminb can take functional constraints, while **nlminb** can only take box constraints.

Furthermore we want to emphasize that **ROI** was built to be extended, as shown in Section 2.7.

2.5.5 ROI read/write

OPs are commonly stored in flat file formats, different solvers allow to read/write different types of this file formats. **ROI** manages the reader/writer registered in the plug-ins, thus allows to write

```
R> lp_file <- tempfile()
R> ROI_write(lp, lp_file, "lp_lpsolve")
R> writeLines(readLines(lp_file))

/* Objective function */
max: +3 C1 +7 C2 -12 C3;

/* Constraints */
+5 C1 +7 C2 +2 C3 <= 61;
+3 C1 +2 C2 -9 C3 <= 35;
+C1 +3 C2 +C3 <= 31;

/* Variable bounds */
-10 <= C3 <= 10;
```

and read OPs in various formats.

```
R> ROI_read(lp_file, "lp_lpsolve")

ROI Optimization Problem:

Maximize a linear objective function of length 3 with
- 3 continuous objective variables,

subject to
- 3 constraints of type linear.
- 1 lower and 1 upper non-standard variable bound.
```

Information about the available reader/writer can be obtained via the functions `ROI_registered_reader()` and `ROI_registered_writer()`.

2.5.6 ROI models

Test problem collections are commonly used in optimization to evaluate and compare the performance of solvers. As each class of optimization problems has its own test sets stored in various formats, **ROI** currently provides access to NETLIB-LP, MIPLIB and the **globalOptTests** package. The NETLIB-LP (Gay 1985) is a collection of linear programming problems, which, even though the main part was created more than 30 years ago is still used today. Mixed integer optimization problems are commonly evaluated using MIPLIB (Koch *et al.* 2011), an extensive collection of academic and industrial MILP applications. The **globalOptTests** (Mullen 2014a) package contains 50 box constrained nonlinear global OPs for benchmarking purposes. These libraries were transformed into **ROI** optimization problems and can be accessed through packages **ROI.models.netlib**, **ROI.models.miplib** (Schwendinger and Theußl 2019) and **ROI.models.globalOptTests** (Schwendinger 2017). Since MIPLIB provides no license file, the OPs are not included in the package but can be easily obtained with the `miplib_download_*()` functions.

```
R> library("ROI.models.miplib")
R> if ( length(miplib()) == 0L ) {
+    miplib_download_benchmark(quiet = TRUE)
+    miplib_download_metainfo()
+  }
R> ops <- miplib("ns1766074")
R> ops

ROI Optimization Problem:

Minimize a linear objective function of length 100 with
- 10 continuous objective variables,
- 90 integer objective variables,

subject to
- 182 constraints of type linear.
- 0 lower and 0 upper non-standard variable bounds.
```

Since the problems are stored as objects of class '`OP`', they can be directly entered into `ROI_solve`.

```
R> library("ROI.models.netlib")
R> agg <- netlib("agg")
R> ROI_solve(agg, "glpk")
```

```
Optimal solution found.
The objective value is: -3.599177e+07
```

ROI makes these data collections (test problem sets) available in a common format, so users can easily compare the different solvers and developers interested in creating optimization software can use them to test their packages. Furthermore, the intuitive structure of **ROI** objects and its use of sparse data structures makes it possible to directly derive a new format for the exchange of linear, quadratic and conic optimization problems.

```
R> library("jsonlite")
R> nested_unclass <- function(x) {
+    x <- unclass(x)
+    if ( is.list(x) )
+      x <- lapply(x, nested_unclass)
+    x
+  }
R> agg_json <- toJSON(nested_unclass(agg))
R> tmp_file <- tempfile()
R> writeLines(agg_json, tmp_file)
```

The resulting text file can be easily imported into any programming language supporting JavaScript Object Notation (JSON). JSON is an open-standard file format that can be parsed by almost all programming languages. For historic reasons, OP collections are commonly provided in flat file formats (e.g., `MPS`, `QPS`). We believe, that today it would be advantageous to store them in general data exchange formats like JSON or XML.

2.5.7 ROI settings

The function `ROI_options()` can be used to set or modify many general and/or solver-related settings. Apart from that **ROI** recognizes some environment variables.

Gradient and Jacobian

When creating a plug-in, the function `G` should be used to derive the gradient and `J` should be used to derive the Jacobian. In **ROI** the gradient and Jacobian are derived analytically for linear and quadratic terms. For the derivation of nonlinear terms, by default, the **numDeriv** (Gilbert and Varadhan 2019) package with the Richardson extrapolation is used. However, this can be easily changed by providing customized functions to derive the gradient or Jacobian function.

```
R> simple_gradient <- function(func, x, ...) {
+    numDeriv::grad(func, x, method = "simple", ...)
+  }
R> ROI_options("gradient", simple_gradient)
R> simple_jacobian <- function(func, x, ...) {
+    numDeriv::jacobian(func, x, method = "simple", ...)
+  }
R> ROI_options("jacobian", simple_jacobian)
```

Solver selection

In the case no solver is provided in `ROI_solve`, the default solver set in `ROI_options` will be used.

```
R> ROI_options("default_solver")
```

```
[1] "auto"
```

By default the option `"default_solver"` is set to `"auto"` which enables automatic solver selection, if any other solver name (e.g., `"glpk"`) is provided the automatic solver selection is discarded in favor of the specified solver.

```
R> ROI_options("default_solver", "glpk")
R> ROI_options("default_solver", "auto")
```

Load plug-ins

The plug-ins are loaded automatically. However, in some situations it is desirable to deactivate the automatic loading and require plug-in packages one at a time. This can be accomplished by setting the environment variable `"ROI_LOAD_PLUGINS"` to `FALSE`.

```
R> Sys.setenv(ROI_LOAD_PLUGINS = FALSE)
```

Afterwards the default load behavior of **ROI** is altered and only the `"nlminb"` solver (which is included in **ROI**) gets registered when `library("ROI")` is called. Therefore, all the other plug-ins have to be loaded manually if needed (e.g., for the `"glpk"` solver one calls `library("ROI.plugin.glpk")`).

2.6 Examples

In this section we provide small examples to introduce the reader into the modeling capabilities of **ROI**.

2.6.1 Linear optimization problems

Consider the LP

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & x_1 + 2x_2 \\ \text{subject to} \quad & x_1 + 8x_2 = 9 \\ & 5x_1 + \ x_2 \leq 6 \\ & x_1 \in [-9, 9], \ x_2 \in [-7, 7]. \end{aligned} \tag{2.19}$$

To solve this LP we use the following **ROI** code:

```
R> lp <- OP(c(1, 2), maximum = TRUE,
+      L_constraint(L = rbind(c(1, 8), c(5, 1)),
+                           dir = c("==", "<="), rhs = c(9, 6)),
+      bounds = V_bound(lb = c(-9, -7), ub = c(9, 7)))
R> (lp_sol <- ROI_solve(lp, "glpk"))

Optimal solution found.
The objective value is: 3.000000e+00

R> solution(lp_sol)

[1] 1 1
```

2.6.2 Quadratic optimization problems

Consider the QP

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2}(x_1^2 + x_2^2) - x_1 \\ \text{subject to} \quad & 4x_1 + 6x_2 \geq 10 \\ & x_1, x_2 \geq 0. \end{aligned} \tag{2.20}$$

Recall that for quadratic terms **ROI** uses the standard form $\frac{1}{2}x^\top Qx + a^\top x$ (see Equation 2.5). Therefore, this problem can be solved by the following **ROI** code:

```
R> qp <- OP(Q_objective(Q = diag(2), L = c(-1, 0)),
+    L_constraint(c(4, 6), ">=", 10))
R> (qp_sol <- ROI_solve(qp, "qpoases"))

Optimal solution found.
The objective value is: -1.538462e-01

R> solution(qp_sol)

[1] 1.4615385 0.6923077
```

To add the constraint $7x_1 + 11x_2 - x_1^2 - x_2^2 \geq 40$ we combine the new constraint with the existing constraint.

```
R> qcqp <- qp
R> constraints(qcqp) <- rbind(constraints(qp),
+      Q_constraint(-diag(c(2, 2)), L = c(7, 11), ">=", 40))
```

In **ROI** a QCQP can be solved by the solvers

```
R> ROI_applicable_solvers(qcqp)

[1] "alabama" "cplex"   "neos"
```

where among these solvers `cplex` and `gurobi` are best suited for this type of problem. However for reproducibility we use the open source solver `alabama`:

```
R> (qcqp_sol <- ROI_solve(qcqp, "alabama", start = c(5, 5)))

Optimal solution found.
The objective value is: 9.447513e+00

R> solution(qcqp_sol)

[1] 2.845720 4.060584
```

and the NEOS server:

```
R> (qcqp_sol <- ROI_solve(qcqp, "neos", method = "mosek"))

Optimal solution found.
The objective value is: 9.447513e+00

R> solution(qcqp_sol)

[1] 2.845695 4.060596
```

2.6.3 Conic optimization problems

Conic optimization problems in standard form (see Equation 2.6) are comprised of a linear objective function and conic constraints. The requirement of a linear objective function is not restrictive, since by making use of the epigraph form (see Equation 2.7) any CP can be transformed into the standard form. In conic optimization conic constraints are used to express predefined linear and nonlinear constraints by the equation $Ax + s = b$, where the slack variable s is required to lie in a specific cone $b - Ax = s \in \mathcal{K}$.

Zero cone

The zero cone is used to model linear equality constraints, since $Ax = b$ is equivalent to $Ax + s = b$, $s \in \mathcal{K}_{\text{zero}}$. The linear constraint $x_1 + 8x_2 = 9 \iff 9 - (1 \;\; 8)x = s \in \mathcal{K}_{\text{zero}}$ can be expressed as follows:

```
R> cpeq <- C_constraint(c(1, 8), K_zero(1), 9)
```

Linear cone

The linear cone is used to model linear less than equal constraints, since $Ax \leq b$ is equivalent to $Ax + s = b$, $s \in \mathcal{K}_{\text{lin}}$. The linear constraint $5x_1 + x_2 \leq 6 \iff 6 - (5 \;\; 1)x = s \in \mathcal{K}_{\text{lin}}$ can be expressed as follows:

```
R> cpleq <- C_constraint(c(5, 1), K_lin(1), 6)
```

By combining the zero cone constraint with the linear cone constraint the LP stated in Equation 2.19 can also be formulated as a CP.

```
R> zlcp <- lp
R> constraints(zlcp) <- c(cpeq, cpleq)
R> (zlcp_sol <- ROI_solve(zlcp, solver = "ecos"))
```

```
Optimal solution found.
The objective value is: 3.000000e+00
```

```
R> solution(zlcp_sol)
```

```
[1] 1 1
```

Note that since in the definition of $\mathcal{K}_{\text{zero}}$ and $\mathcal{K}_{\text{lin}}$ only one variable is involved, a single zero cone or linear cone constraint can be expressed by a single row of the constraint matrix A. For all the other cones at least two rows of the of the constraint matrix A will be needed to express a single conic constraint.

Second-order cone

The second-order cone is used to express constraints of the type $||u||_2 \leq w$ (see Equation 2.9), where the variables $u \in \mathbb{R}^{n-1}$ and $w \in \mathbb{R}$ are expressed by $b - Ax \in \mathcal{K}^n_{\text{soc}}$. Specifically, w is expressed by $b_1 - a_1^\top x$ and u_i is expressed by $b_i - a_i^\top x$, $i \in 2, .., n$.

Consider the SOCP

$$\begin{aligned} \underset{(y,t)}{\text{maximize}} \quad & y_1 + y_2 \\ \text{subject to} \quad & \sqrt{(2 + 3y_1)^2 + (4 + 5y_2)^2} \leq 6 + 7t \\ & y_1,\ y_2 \in \mathbb{R},\ t \in (-\infty, 9] \end{aligned} \tag{2.21}$$

for $x = (y_1, y_2, t)^\top$, the constraint

$$\sqrt{(2 + 3y_1)^2 + (4 + 5y_2)^2} \leq 6 + 7t$$

is equivalent to

$$\sqrt{(b_2 - a_2^\top x)^2 + (b_3 - a_3^\top x)^2} \leq b_1 - a_1^\top x,$$

where

$$A = \begin{pmatrix} 0 & 0 & -7 \\ -3 & 0 & 0 \\ 0 & -5 & 0 \end{pmatrix}, \; b = \begin{pmatrix} 6 \\ 2 \\ 4 \end{pmatrix}. \tag{2.22}$$

Given the constraint matrix A and the right hand side b this can be directly translated into the following OP:

```
R> soc1 <- OP(c(1, 1, 0),
+    C_constraint(L = rbind(c(0, 0, -7), c(-3, 0, 0), c(0, -5, 0)),
+      cone = K_soc(3), rhs = c(6, 2, 4)), maximum = TRUE,
+    bounds = V_bound(ld = -Inf, ui = 3, ub = 9, nobj = 3))
R> (soc1_sol <- ROI_solve(soc1, solver = "ecos"))

Optimal solution found.
The objective value is: 2.535571e+01

R> solution(soc1_sol)

[1] 19.055671  6.300041  9.000000
```

Consider the SOCP

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \sqrt{x_1^2 + x_2^2} \\ \text{subject to} \quad & x_1 + x_2 = 2 \\ & x_1, \; x_2 \geq 0 \end{aligned} \tag{2.23}$$

by making use of the epigraph form the OP can rewritten into the standard form.

$$\begin{aligned} \underset{(x,t)}{\text{minimize}} \quad & t \\ \text{subject to} \quad & \sqrt{x_1^2 + x_2^2} \leq t \\ & x_1 + x_2 = 2 \\ & x_1, \; x_2 \geq 0 \end{aligned} \tag{2.24}$$

This problem can be solved by the following **ROI** code:

```
R> A <- rbind(c(0, 0, -1), c(-1, 0, 0), c(0, -1, 0))
R> soc2 <- OP(objective = L_objective(c(0, 0, 1)),
```

```
+    constraints = c(C_constraint(A, c(K_soc(3)), c(0, 0, 0)),
+      L_constraint(c(1, 1, 0), "==", 2)))
R> (soc2_sol <- ROI_solve(soc2, solver = "ecos"))

Optimal solution found.
The objective value is: 1.414214e+00

R> solution(soc2_sol)

[1] 1.000000 1.000000 1.414214
```

Exponential cone

The primal exponential cone is used to express constraints of the type $ve^{\frac{u}{v}} \leq w$ (see Equation 2.11), here $u \in \mathbb{R}$, $v \in \mathbb{R}$, $w \in \mathbb{R}$ and $v > 0$. Since three scalar variables are evolved one primal exponential cone adds three rows to the constraint matrix A. The variables u, v and w are again expressed by the corresponding elements of $b - Ax$. Specifically, $u := b_1 - a_1^\top x$, $v := b_2 - a_2^\top x$ and $w := b_3 - a_3^\top x$.

Consider the CP

$$\begin{aligned} \underset{(y,t)}{\text{maximize}} \quad & y_1 + 2y_2 \\ \text{subject to} \quad & \exp(7 + 3y_1 + 5y_2) \leq 9 + 11t_1 + 12t_2 \\ & y_1,\ y_2 \in (-\infty, 20],\ t_1,\ t_2 \in (-\infty, 50] \end{aligned} \tag{2.25}$$

the constraint $\exp(7 + 3y_1 + 5y_2) \leq 9 + 11t_1 + 12t_2$ can be represented by the exponential cone by recognizing that $u = 7+3y_1+5y_2$, $v = 1$ and $w = 9+11t_1+12t_2$.

For $x = (y_1, y_2, t_1, t_2)^\top$ this leads to the matrices

$$A = \begin{pmatrix} -3 & -5 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -11 & -12 \end{pmatrix},\ b = \begin{pmatrix} 7 \\ 1 \\ 9 \end{pmatrix}, \tag{2.26}$$

and the following conic constraint:

```
R> cexpp <- C_constraint(L = rbind(c(-3, -5, 0, 0), c(0, 0, 0, 0),
+    c(0, 0, -11, -12)), cone = K_expp(1), rhs = c(7, 1, 9))
```

Therefore this CP can be solved by the following **ROI** code:

```
R> expp1 <- OP(c(1, 2, 0, 0), cexpp, maximum = TRUE,
+      bounds = V_bound(ld = -Inf, ub = c(20, 20, 50, 50)))
R> (expp1_sol <- ROI_solve(expp1, solver = "ecos"))
```

```
Optimal solution found.
The objective value is: 6.685104e+00
```

```
R> solution(expp1_sol)
```

```
[1] -33.31490  20.00000  49.99996  49.99996
```

In many statistical models the objective contains logarithmic and exponential terms (e.g., logistic regression, relative risk regression, Poisson regression). As a simple example consider the CP

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \log(9 + 7x) \\ \text{subject to} \quad & 0 \leq x \leq 1, \end{aligned} \tag{2.27}$$

or in the epigraph form

$$\begin{aligned} \underset{(x,t)}{\text{maximize}} \quad & t \\ \text{subject to} \quad & \log(9 + 7x) \geq t \\ & 0 \leq x \leq 1. \end{aligned} \tag{2.28}$$

Taking the exponential of $\log(w) \geq u$ gives $w \geq \exp(u)$, therefore the constraint $\log(9 + 7x) \geq t$ can be represented by the exponential cone by recognizing that $u = t$, $v = 1$ and $w = 9 + 7x$, which yields the matrices

$$A = \begin{pmatrix} 0 & -1 \\ 0 & 0 \\ -7 & 0 \end{pmatrix}, \; b = \begin{pmatrix} 0 \\ 1 \\ 9 \end{pmatrix}. \tag{2.29}$$

Therefore this CP can be solved by the following **ROI** code:

```
R> A <- rbind(c(0, -1), c(0, 0), c(-7, 0))
R> log1 <- OP(L_objective(c(0, 1), c("x", "t")),
+           C_constraint(A, K_expp(1), rhs = c(0, 1, 9)),
+           bounds = V_bound(lb = c(0, -Inf), ub = c(1, Inf)),
+           maximum = TRUE)
R> (log1_sol <- ROI_solve(log1, solver = "ecos"))
```

```
Optimal solution found.
The objective value is: 2.772589e+00
```

```
R> solution(log1_sol)
```

```
       x        t
1.000000 2.772589
```

Power cone

The primal power cone is used to express constraints of the type $u^\alpha v^{1-\alpha} \geq |w|$ (see Equation 2.13), here $u \in \mathbb{R}$, $v \in \mathbb{R}$, $w \in \mathbb{R}$, $u, v \geq 0$ and $\alpha \in [0, 1]$. Since three scalar variables are involved, one primal exponential cone adds three rows to the constraint matrix A. The variables u, v and w are again expressed by the corresponding elements of $b - Ax$. Specifically $u := b_1 - a_1^\top x$, $v := b_2 - a_2^\top x$ and $w := b_3 - a_3^\top x$.

Consider the CP

$$\begin{aligned} \underset{y}{\text{minimize}} \quad & 3y_1 + 5y_2 \\ \text{subject to} \quad & 5 + y_1 \geq (2 + y_2)^4 \\ & y_1 \geq 0, \; y_2 \geq 2. \end{aligned} \tag{2.30}$$

The constraint $5 + y_1 \geq (2 + y_2)^4$ can be represented by the power cone by recognizing that $u = 5 + y_1$, $v = 1$, $w = 2 + y_2$ and $\alpha = \frac{1}{4}$. For $x = (y_1, y_2)^\top$ this leads to the matrices

$$A = \begin{pmatrix} -1 & 0 \\ 0 & 0 \\ 0 & -1 \end{pmatrix}, b = \begin{pmatrix} 5 \\ 1 \\ 2 \end{pmatrix}, \tag{2.31}$$

and the following conic constraint:

```
R> A <- rbind(c(-1, 0), c(0, 0), c(0, -1))
R> cpowp <- C_constraint(A, K_powp(1/4), rhs = c(5, 1, 2))
```

Therefore, this CP can be solved by the following **ROI** code:

```
R> powp1 <- OP(c(3, 5), cpowp, bounds = V_bound(lb = c(0, 2)))
R> (powp1_sol <- ROI_solve(powp1, solver = "scs", max_iter = 1e6))

Optimal solution found.
The objective value is: 7.630000e+02

R> solution(powp1_sol)

[1] 250.999986   2.000001
```

Positive semidefinite cone

The positive semidefinite cone is used to express constraints of the type $X \in \mathcal{S}^n$ and $z^\top X z \geq 0$ for all $z \in \mathbb{R}^n$ (see Equation 2.10). Positive semidefinite constraints are expressed by the constraint matrix A and right hand side b. To express the linear matrix inequality $\sum_{i=1}^n x_i F_i \preceq F_0$ which is equivalent to $F_0 - \sum_{i=1}^n x_i F_i \in \mathcal{K}_{psd}^d$, in terms of $b - Ax \in \mathcal{K}_{psd}$ the symmetric matrices $F_i \in R^{d \times d}$ are transformed into vectors by a half-vectorization. Half-vectorization is a special kind of matrix vectorization for symmetric matrices, which transforms a symmetric matrix

```
R> (A <- matrix(c(1, 2, 3, 2, 4, 5, 3, 5, 6), nrow = 3))
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    4    5
[3,]    3    5    6
```

into a vector. The function `vech` correspondingly transforms n symmetric $d \times d$ matrices into a $(d(d+1)/2) \times n$ matrix:

```
R> vech(A)
```

```
     [,1]
[1,]    1
[2,]    2
[3,]    3
[4,]    4
[5,]    5
[6,]    6
```

Consider the SDP

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & x_1 + x_2 - x_3 \\ \text{subject to} \quad & x_1 \begin{pmatrix} 10 & 3 \\ 3 & 10 \end{pmatrix} + x_2 \begin{pmatrix} 6 & -4 \\ -4 & 10 \end{pmatrix} + x_3 \begin{pmatrix} 8 & 1 \\ 1 & 6 \end{pmatrix} \preceq \begin{pmatrix} 16 & -13 \\ -13 & 60 \end{pmatrix} \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

This problem can be solved by the following **ROI** code:

```
R> F1 <- rbind(c(10, 3), c(3, 10))
R> F2 <- rbind(c(6, -4), c(-4, 10))
R> F3 <- rbind(c(8, 1), c(1, 6))
R> F0 <- rbind(c(16, -13), c(-13, 60))
R> psd <- OP(objective = L_objective(c(1, 1, -1)),
+            constraints = C_constraint(L = vech(F1, F2, F3),
+                                       cone = K_psd(3), rhs = vech(F0)))
R> (psd_sol <- ROI_solve(psd, solver = "scs"))
```

```
Optimal solution found.
The objective value is: -1.486458e+00
```

```
R> solution(powp1_sol)
```

```
[1] 250.999986   2.000001
```

2.6.4 General nonlinear optimization problems

The following example from Rosenbrock (1960) is known as Rosenbrock's post office problem.

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & x_1\ x_2\ x_3 \\ \text{subject to} \quad & x_1 + 2x_2 + 2x_3 \leq 72 \\ & x_1,\ x_2,\ x_3 \in [0, 42] \end{aligned}$$

```
R> nlp_1 <- OP(maximum = TRUE, bounds = V_bound(ud = 42, nobj = 3L))
R> objective(nlp_1) <- F_objective(F = function(x) prod(x), n = 3,
+    G = function(x) c(prod(x[-1]), prod(x[-2]), prod(x[-3])))
R> constraint <- function(x) x[1] + 2 * x[2] + 2 * x[3]
R> constraints(nlp_1) <- F_constraint(F = constraint, dir = "<=",
+                         rhs = 72, J = function(x) c(1, 2, 2))
R> nlp_1

ROI Optimization Problem:

Maximize a nonlinear objective function of length 3 with
- 3 continuous objective variables,

subject to
- 1 constraint of type nonlinear.
- 0 lower and 3 upper non-standard variable bounds.
```

Alternatively the linear constraint $x_1 + 2x_2 + 2x_3 \leq 72$ could and should be modeled directly as a linear constraint,

```
R> nlp_2 <- nlp_1
R> constraints(nlp_2) <- L_constraint(L = c(1, 2, 3), "<=", 72)
R> nlp_2

ROI Optimization Problem:

Maximize a nonlinear objective function of length 3 with
- 3 continuous objective variables,

subject to
- 1 constraint of type linear.
- 0 lower and 3 upper non-standard variable bounds.
```

using `L` and `Q` constraints rather than `F_constraint` has the advantage that for `L` and `Q` constraints the Jacobian is derived analytically if needed and not provided.

In contrast to LP, QP and CP solvers almost all GPS require that users specify starting values. The choice of the starting values has a big influence on how fast and to which solution the algorithm will converge.

```
R> (nlp_1_s_1 <- ROI_solve(nlp_1, "alabama", start = c(10, 10, 10)))

Optimal solution found.
The objective value is: 3.456000e+03

R> solution(nlp_1_s_1)

[1] 24.00002 11.99999 11.99999

R> (nlp_1_s_2 <- ROI_solve(nlp_1, "alabama", start = c(20, 20, 20)))

No optimal solution found.
The solver message was: Convergence due to lack of progress in
parameter updates.
The objective value is: 1.314286e+308

R> solution(nlp_1_s_2)

[1] NA NA NA
```

There are several possibilities to help the algorithm to find a good solution. In almost all practical applications it is possible to specify lower and upper bounds. Carefully chosen bounds can improve the quality of the solution and decrease the runtime of the algorithm. The runtime of the algorithm also strongly depends on the tolerances set, if the tolerances are set to small the algorithm will reach the maximum number of iterations before convergence.

```
R> (nlp_1_s_3 <- ROI_solve(nlp_1, "alabama", start = c(10, 10, 10),
+    tol = 1E-24))

No optimal solution found.
The solver message was: ALABaMA ran out of iterations and did not
converge.
The objective value is: 3.456000e+03

R> solution(nlp_1_s_3, force = TRUE)

[1] 24.00002 11.99999 11.99999
```

Last but not least the chosen method has a big influence on the solution. As mentioned before some algorithms are designed to search for a global solution others are designed to search for a local solution. **ROI.plugin.deoptim** provides access to the packages **DEoptim** (Mullen, Ardia, Gil, Windover, and Cline 2011) and **DEoptimR** which implement a differential evolution algorithm for global optimization. For more information about continuous global optimization in R we refer to Mullen (2014a).

```
R> (nlp_1_s_4 <- ROI_solve(nlp_1, "deoptimr", start = c(20, 20, 20),
+    max_iter = 400, tol = 1E-6))

Optimal solution found.
The objective value is: 3.456000e+03

R> solution(nlp_1_s_4)

[1] 23.99990 12.00002 12.00003
```

In general it is often hard to obtain a global optimum for non-convex optimization problems. Often the only option is to try different algorithms, parameters and starting values and hope that one of the solutions is a global optimum. **ROI** lowers the burden to compare different algorithms and therefore can assist in finding a global solution.

2.6.5 Mixed integer problems

Consider the MIP

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & 5x_1 + 7x_2 \\ \text{subject to} \quad & 5x_1 + 3x_2 \geq 7 \\ & 7x_1 + 1x_2 \geq 9 \\ & x_1 \in \{0, 1\},\ x_2 \geq 0, x_2 \in \mathbb{Z} \end{aligned} \tag{2.32}$$

this problem can be solved by the following **ROI** code:

```
R> A <- rbind(c(5, 3), c(7, 1))
R> milp <- OP(c(5, 7),
+    constraints = L_constraint(A, c(">=", ">="), c(7, 9)),
+    types = c("B", "I"))
R> (milp_sol <- ROI_solve(milp, solver = "glpk"))

Optimal solution found.
The objective value is: 1.900000e+01

R> solution(milp_sol)

[1] 1 2
```

2.7 Extending ROI

To stay abreast of changes and to further the availability of different solvers in the **ROI** ecosystem, **ROI** allows developers to integrate their own extensions, so called plug-ins. This can be seen as a key feature, since it allows the use of new solvers with no or minimal code changes.

Extending **ROI** with a new solver method can be split into three parts. First, a function to be called by **ROI** has to be written. Second, the function plus information about the function are added into the **ROI** solver registry. Third, a mapping from the solver specific arguments and the status codes to their **ROI** counterpart has to be provided.

2.7.1 Signatures

In order to establish a connection between the OP and the solvers provided via plug-ins, both are equipped with a signature. The signature captures all the information necessary to determine which solver is applicable to a given problem.

```
R> OP_signature(lp)

  objective constraints bounds cones maximum    C     I     B
1         L           L      V     X    TRUE TRUE FALSE FALSE
```

New signatures are created by the function `ROI_plugin_make_signature()`. The following shows how to create the signature for the glpk solver,

```
R> glpk_signature <- ROI_plugin_make_signature(objective = "L",
+    constraints = "L", types = c("C", "I", "B", "CI", "CB", "IB",
+    "CIB"), bounds = c("X", "V"), maximum = c(TRUE, FALSE))
```

where the objective and the constraints have to be linear. Furthermore, this signature indicates that, the variable types are allowed to be binary (`"B"`), integer (`"I"`), continuous (`"C"`) or any combinations of them. The bounds have to be variable bounds (`"V"`) or no bounds at all encoded by `"X"`. The last argument maximum specifies that GLPK can find both maxima and minima.

2.7.2 Writing a new solver method

Any function supposed to add a solver to **ROI** has to take the arguments `x` and `control`, where `x` is of class '`OP`' and `control` a list containing the additional arguments. Furthermore, the solution has to be canonicalized before it is returned. The following shows the code from **ROI.plugin.glpk** for solving linear problems.

```
R> glpk_solve_OP <- function(x, control = list()) {
+    control$canonicalize_status <- FALSE
+    glpk <- list(Rglpk_solve_LP, obj = terms(objective(x))[["L"]],
+      mat = constraints(x)$L, dir = constraints(x)$dir,
+      rhs = constraints(x)$rhs, bounds = bounds(x),
+      types = types(x), max = maximum(x), control = control)
+    mode(glpk) <- "call"
+    if ( isTRUE(control$dry_run) )
+      return(glpk)
+
+    out <- eval(glpk)
+    ROI_plugin_canonicalize_solution(solution = out$solution,
+      optimum = out$optimum, status = out$status, solver = "glpk",
+      message = out)
+  }
```

As can be seen from this example, most plug-ins support the optional control argument `dry_run`, which returns the solver call. This is especially useful for debugging wrapper functions with more transformation steps, so the data used in the solver call can be easily shared and inspected.

2.7.3 Register solver methods

Registering a solver method can be seen as telling **ROI** which function it should use when `ROI_solve()` with argument `solver` set to the name of the plug-in (e.g., `"glpk"`) is called. In order to avoid ambiguity, each plug-in should at most provide one method for each problem type. Solver methods are registered via the function `ROI_plugin_register_solver_method()`, which takes as arguments the problem types (as signatures), the solver name and a wrapper function `ROI_solve()` is dispatched to.

```
ROI_plugin_register_solver_method(glpk_signature, "glpk",
                                  glpk_solve_OP)
```

After the solver registration the name of the solver will appear among the registered solvers.

2.7.4 Adding additional information

To be able to provide a consistent interface, each plug-in has to define a mapping between the solver specific status codes and the status codes used by **ROI**, as well as a mapping between solver specific control variables and **ROI** control variables.

Status codes

Status codes are added via the function `ROI_plugin_add_status_code_to_db()`:

```
ROI_plugin_add_status_code_to_db(solver = "glpk", code = 5L,
  symbol = "GLP_OPT",
  message = "Solution is optimal.",
  roi_code = 0L)
```

Here, the `"glpk"` specific status code `5L` is mapped to the canonicalized **ROI** status code `0L`, which signals that the solution is optimal as indicated by the status message.

Control variables

Plug-ins are contracted to provide a mapping between the names of the control variables used by **ROI** and the names of the control variables used by the plug-in. The following maps the **glpk** argument `tm_limit` to the **ROI** equivalent `max_time`.

```
ROI_plugin_register_solver_control(solver = "glpk",
  args = "tm_limit", roi_control = "max_time")
```

2.7.5 Register reformulations

While in Section 2.5.3 we showed how to use reformulations, here we explain how new reformulations can be added through plug-ins. Again, the signature is used to define which transformations can be performed by a given method.

```
R> bqp_signature <- ROI_plugin_make_signature(objective = "Q",
+    constraints = c("X", "L"), types = "B", bounds = c("X", "V"),
+    cones = c("X"), maximum = c(TRUE, FALSE))
R> milp_signature <- ROI_plugin_make_signature(objective = "L",
+    constraints = c("X", "L"),
+    types = c("C", "I", "B", "CI", "CB", "IB", "CIB"),
+    bounds = c("X", "V"), maximum = c(TRUE, FALSE), cones = c("X"))

ROI_plugin_register_reformulation( from = bqp_signature,
  to = milp_signature, method_name = "bqp_to_lp",
  method = bqp_to_lp, description = "", cite = "", author = "")
```

The code above registers the function `bqp_to_lp()`, which is based on the function `.linearize_BQP()` from the **relations** package, as a new reformulation named `"bqp_to_lp"`. The parameter `from` defines which signatures the original problem is allowed to have and `to` defines all possible signatures the reformulation could have.

2.7.6 Register reader/writer

Plug-ins can also add new read and write functions. Any method to be registered as read function has to take as arguments the file name `file` and `...` for optional additional arguments.

```
R> library("slam")
R> json_reader_lp <- function(file, ...) {
+    stopifnot(is.character(file))
+    y <- read_json(file, simplifyVector = TRUE)
+    to_slam <- function(x) do.call(simple_triplet_matrix, x)
+    x <- OP()
+    objective(x) <- L_objective(
+        to_slam(y[["objective"]][["L"]]),
+        y[["objective"]][["names"]])
+    constraints(x) <- L_constraint(
+        to_slam(y[["constraints"]][["L"]]),
+        y[["constraints"]][["dir"]], y[["constraints"]][["rhs"]],
+        y[["constraints"]][["names"]])
+    types(x) <- y[["types"]]
+    bounds(x) <- structure(y[["bounds"]],
+                           class = c("V_bound", "bound"))
+    maximum(x) <- as.logical(y[["maximum"]])
+    x
+  }
```

The write functions need the additional argument `x`, which is the OP to be written.

```
R> json_writer_lp <- function(x, file, ...) {
+    writeLines(toJSON(nested_unclass(x), null = "null"),
+               con = file)
+  }
```

Using the JSON based exchange format proposed in Section 2.5.6, we illustrate how to register simple JSON read and write functions for linear problems.

```
R> plugin_name <- "io"
R> ROI_plugin_register_writer("json", plugin_name, milp_signature,
+    json_writer_lp)
R> ROI_plugin_register_reader("json", plugin_name, json_reader_lp)
```

After the registration of the functions they can be used in the typical way.

```
R> fname <- tempfile()
R> file <- ROI_write(lp, file = fname, type = "json")
R> (lp_json <- ROI_read(file = fname, type = "json"))

ROI Optimization Problem:

Maximize a linear objective function of length 2 with
- 2 continuous objective variables,

subject to
- 2 constraints of type linear.
- 2 lower and 2 upper non-standard variable bounds.
```

2.7.7 ROI tests

Writing tests is an important task in software development. The **ROI.tests** package provides a collection of tests which should be applied to any **ROI** plug-in during development. Since **ROI** knows the signature of each solver, **ROI.tests** can select the appropriate tests based on the solver name.

```
R> library("ROI.tests")
R> test_solver("glpk")

LP-01:  OK!
LP-02:  OK!
LP-03:  OK!
MILP-01:
ERROR in MILP-01@01
  OK!
MILP-02:  OK!
```

2.8 Applications

In the following we demonstrate how **ROI** can be used to solve selected problems from statistics, namely L_1 regression, best subset selection, relative risk regression, sum-of-norms clustering, and graphical lasso.

2.8.1 L_1 regression

The linear programming formulation of the L_1 regression problem as shown in Section 2.2.1 can be constructed using **ROI** methods via the following R function.

```
R> create_L1_problem <- function(x, j) {
+    m <- ncol(x) + 1L
+    n <- 2 * nrow(x)
+    beta <- double(m + n)
+    beta[j + 1] <- 1
+    OP(objective = L_objective(c(rep(0, m), rep(1, n))),
+      constraints = rbind(
+        L_constraint(L = cbind(1, x, diag(nrow(x)),
+                               -diag(nrow(x))),
+          dir = eq(nrow(x)), rhs = rep(0, nrow(x))),
+        L_constraint(L = beta, dir = "==", rhs = -1)),
+      bounds = V_bound(li = seq_len(m), lb = rep(-Inf, m),
+        nobj = length(beta)))
+  }
```

One can solve e.g., Brownlee's stack loss plant data example from the **stats** package using the above OP and the solver GLPK as follows.

```
R> data("stackloss")
R> l1p <- create_L1_problem(as.matrix(stackloss), 4)
R> L1_res <- ROI_solve(l1p, solver = "glpk")
R> solution(L1_res)[1:ncol(stackloss)]
```

```
[1] -39.68985507   0.83188406   0.57391304  -0.06086957
```

The first value corresponds to the intercept and the others to the model coefficients.

2.8.2 Best subset selection

Recently Bertsimas *et al.* (2016) reported a bewildering 450 billion factor speedup from 1991 to 2015 for solving MIP, which is partly due to algorithmic improvements and partly through hardware speedups. They show how this speed gain can be utilized to solve the best subset selection problem in regression (see for example Miller 2002), which is an NP-hard combinatorial OP. The best subset selection problem is a variable selection scheme which extends linear least-squares by adding a constraint on the number of predictor variables.

$$\underset{\beta}{\text{minimize}}\ \frac{1}{2}||y - X\beta||_2^2 \quad \text{subject to} \quad \sum_{i=1}^{p} \mathbb{I}_{\{\beta_i \neq 0\}} \leq k \tag{2.33}$$

As Equation 2.33 suggests, the best subset selection is in spirit similar to ridge regression and lasso. However instead of the l_2 and l_1 norm best subset selection

uses the l_0 norm which makes it non-convex and therefore hard to solve. The **leaps** (Lumley 2017) package implements an efficient branch-and-bound algorithm which is significantly faster than exhaustive search. Using a optimization solver has the additional advantage that it is possible to impose additional restrictions, e.g., if the quadratic term of a covariate is selected to be in the equation the linear term has also to be selected. In **ROI** best subset selection can be either implemented as mixed integer quadratic problem or as mixed integer second order cone problem. This problem can be solved with a mixed integer QP solver or a mixed integer SCOP solver. An implementation of the second order cone version can be found in the Appendix B.1.

2.8.3 Relative risk regression

Generalized linear models (GLM) are often the statisticians' first choice for regression analysis of binary response data. The most prominent model of the GLM family is logistic regression[3] A GLM which is mainly used in epidemiology but closely related to logistic regression is relative risk regression. The main difference between these two models is the fact that relative risk regression uses the log link and therefore estimates relative risks instead of the odds ratio. Lumley, Kronmal, and Ma (2006) reviews the algorithms purposed in the literature to perform relative risks regression. Luo, Zhang, and Sun (2014) suggest to use a log-binomial model with

$$\underset{\beta}{\text{maximize}} \sum_{i=1}^{n} y_i \, X_{i*}\beta + \sum_{i=1}^{n} (1 - y_i) \, \log(1 - \exp(X_{i*}\beta)) \quad \text{subject to} \quad X\beta \leq 0. \tag{2.34}$$

Here X_{i*} refers to the i-th row of the data matrix X, the constraint $X\beta \leq 0$ ensures that the estimated probabilities are in the interval [0, 1]. The log-binomial regression model can be formulated as a general nonlinear optimization problem

```
R> logbin_gps <- function(y, X) {
+    loglikelihood <- function(beta) {
+      xb <- drop(X %*% beta)
+      if (any(xb > 0)) NaN
+      else sum(y * xb + (1 - y) * log(1 - exp(xb)))
+    }
+
+    gradient <- function(beta) {
+      exb <- exp(drop(X %*% beta))
+      drop(crossprod(X, (y - exb) / (1 - exb)))
```

[3] An example for logistic regression can be found on the **ROI** homepage.

```
+    }
+
+    OP(F_objective(loglikelihood, n = ncol(X), G = gradient),
+      L_constraint(L = X, dir = leq(nrow(X)),
+                   rhs = double(nrow(X))),
+      bounds = V_bound(ld = -Inf, nobj = ncol(X)), maximum = TRUE)
+  }
```

which can be solved by any GPS which allows to specify linear constraints (e.g., **alabama**). Alternatively the log-binomial regression model can be solved by any CP solver which supports the linear and the primal exponential cone (e.g., **scs**). This formulation has attractive theoretical and numerical convergence guarantees without the need to find suitable starting values.

```
R> logbin_cp <- function(y, X, rhs_eps = 1e-7) {
+    y_is_0 <- y == 0L
+    n_y_is_0 <- sum(y_is_0)
+    o <- OP(c(y %*% X, double(n_y_is_0), rep(1, n_y_is_0)),
+            maximum = TRUE)
+    L1 <- cbind(X, matrix(0, nrow(X), 2 * n_y_is_0))
+    log1exp <- function(xi, j, n_y_is_0) {
+      M <- matrix(0, nrow = 6, ncol = length(xi) + 2 * n_y_is_0)
+      M[1, seq_along(xi)] <- -xi
+      M[3, length(xi) + j] <- -1
+      M[4, length(xi) + n_y_is_0 + j] <- -1
+      M[6, length(xi) + j] <- 1
+      M
+    }
+    L2 <- mapply(log1exp, split(X[y_is_0,], seq_len(n_y_is_0)),
+      seq_len(n_y_is_0), MoreArgs = list(n_y_is_0 = n_y_is_0),
+      SIMPLIFY = FALSE)
+    rhs <- c(c(0, 1, 0), c(0, 1, 1))
+    rhs <- c(rep(-rhs_eps, nrow(X)), rep(rhs, n_y_is_0))
+    cones <- c(K_lin(nrow(X)), K_expp(2 * n_y_is_0))
+    L <- do.call(rbind, c(list(L1), L2))
+    constraints(o) <- C_constraint(L, cones, rhs)
+    bounds(o) <- V_bound(ld = -Inf, nobj = length(objective(o)))
+    o
+  }
```

To illustrate the estimation of GLMs with binary responses using **ROI**, we generate a data set similar to an example in the **rms** (Harrell Jr 2019) manual.

```
R> generate_data <- function(n) {
+    treat <- factor(sample(c('a','b','c'), n, TRUE))
+    num.diseases <- sample(0:4, n, TRUE)
+    age <- rnorm(n, 50L, 10L)
+    cholesterol <- rnorm(n, 200L, 25L)
+    weight <- rnorm(n, 150L, 20L)
+    sex <- factor(sample(c('female','male'), n, TRUE))
+
+    # Specify population model for log odds that Y = 1
+    L <- ( -1 + 0.1 * (num.diseases - 2) + 0.045 * (age - 70)
+      + (log(cholesterol - 10) - 5.2) - 2 * (treat == 'a')
+      + 0.5 * (treat == 'b') - 0.5 * (treat == 'c') )
+    # Simulate binary y to have Prob(y = 1) = 1 / (1 + exp(-L))
+    y <- as.double(runif(n) < exp(L))
+
+    A <- cbind(intercept = 1, age, sex, weight,
+      logchol = log(cholesterol - 10), num.diseases,
+      treatb = (treat == "b"), treatc = (treat == "c"))
+    return(list(y = y, A = A))
+  }

R> set.seed(1234)
R> dat <- generate_data(1500L)

R> start <- c(log(0.2), double(ncol(dat$A) - 1))
R> prob_login_bin_gps <- logbin_gps(dat$y, dat$A)
R> s1 <- ROI_solve(prob_login_bin_gps, "alabama", start = start)
R> solution(s1)

[1] -1.312349e+01  3.673749e-02  9.992758e-03 -5.263405e-04
[5]  1.420533e+00  3.308212e-02  2.631972e+00  1.271524e+00

R> prob_login_bin_cp <- logbin_cp(dat$y, dat$A)
R> s2 <- ROI_solve(prob_login_bin_cp, solver = "ecos")
R> head(solution(s2), ncol(dat$A))

[1] -1.309742e+01  3.674558e-02  1.007736e-02 -5.345493e-04
[5]  1.414958e+00  3.307946e-02  2.636203e+00  1.275390e+00
```

```
R> obj_fun <- objective(prob_login_bin_gps)
R> obj_fun(head(solution(s2), ncol(dat$A))) - obj_fun(solution(s1))
```

```
[1] 0.0001827799
```

We see that both approaches yield a similar result.

2.8.4 Sum-of-norms clustering

Borrowing ideas from regularization, sum-of-norms (SON) clustering (convex clustering) is an interesting alternative to established clustering approaches like hierarchical or k-means clustering, which has attracted a lot of research in recent years (Pelckmans, De Brabanter, De Moor, and Suykens 2005; Lindsten, Ohlsson, and Ljung 2011; Hocking, Joulin, Bach, and Vert 2011; Zhu, Xu, Leng, and Yan 2014; Chi and Lange 2015; Tan, Witten, and others 2015). Pelckmans *et al.* (2005); Hocking *et al.* (2011) describe SON clustering as a convexification of hierarchical clustering and Lindsten *et al.* (2011) establish that SON clustering can be seen as a convex relaxation of k-means clustering. Due to its convexity, SON clustering is not dependent on the starting values, which is a clear advantage over the non-convex k-means and hierarchical clustering.

SON clustering solves the following convex OP,

$$\underset{M_i}{\text{minimize}} \ \frac{1}{2}\sum_{i=1}^{m} ||X_{i*} - M_{i*}||_2^2 + \lambda \sum_{i<j} ||M_{i*} - M_{j*}||_q \tag{2.35}$$

where $q \in \{1, 2, \infty\}$, $X \in \mathbb{R}^{m \times n}$ is the data matrix and M_{i*} the i-th row of the optimization variable M. The regularization term $\lambda \sum_{i<j} ||M_{i*} - M_{j*}||_q$ induces equal rows M_{i*}. For $\lambda = 0$ all rows are unique and M is equal to X, when λ increases the number of unique rows of M will decrease. This gives a clustering where all equal rows belong to the same cluster. By solving Equation 2.35 for different λ_i, where $\lambda_1 < \lambda_2 < \cdots < \lambda_{k-1} < \lambda_k$, one can obtain a hierarchical clustering tree (Pelckmans *et al.* 2005).

At least two implementations of SON clustering exist in R, Chi and Lange (2015) provide a fast implementation of SON clustering on CRAN (`https://cran.r-project.org/package=cvxclustr`) and Hocking *et al.* (2011) provide their code on R-Forge (`https://r-forge.r-project.org/projects/clusterpath/`).

A **ROI** formulation as SOCP of SON clustering can be found in the Appendix B.2.

2.8.5 Graphical lasso

Obtaining good estimates of the covariance matrix Σ is important in modern statistics. Often Σ is not estimated directly but its inverse, the precision matrix $\Theta = \Sigma^{-1}$

is (e.g., Meinshausen and Bühlmann (2006)). Estimating the precision matrix instead of the covariance matrix has the advantage that there is a direct connection between the precision matrix and Gaussian graphical models, in the sense that the precision matrix defines the structure of the Gaussian graphical model. Since the elements of the precision matrix are the partial correlations, Θ_{ij} is zero if and only if i and j are conditionally independent. Translated to Gaussian graphical models, two edges A and B are only connected if the corresponding entry in the precision matrix is non zero (Lauritzen 1996).

Several authors proposed an algorithm connected to the lasso (Tibshirani 1996), to obtain a sparse estimate of the precision matrix, the so-called graphical lasso (glasso) (Friedman, Hastie, and Tibshirani 2008). The glasso solves the following convex OP,

$$\underset{\Theta \succ 0}{\text{minimize}} \quad -\log(\det(\Theta)) + \text{tr}(S\ \Theta) + \lambda\ ||X||_1 \tag{2.36}$$

where the data matrix $X \in \mathbb{R}^{m \times p}$ is assumed to be generated from a p-dimensional multivariate normal distribution $N_p(\mu, \Sigma)$ and S is the sample covariance matrix of X. Making use of the exponential and semidefinite cone, this can be brought into the CP standard form and solved by **ROI** using **SCS**. The corresponding R code can be found in the Appendix B.3.

2.9 Conclusions

In the paper we presented the **ROI** package and its extensions. **ROI** provides a consistent way to model OPs in R. **ROI** makes strong use of R's generic functions, such that users already familiar with R are not obliged to learn a new language. The plug-in packages equip **ROI** with optimization solvers and predefined optimization models. **ROI** is currently applicable to linear, quadratic, conic and general nonlinear OPs and provides access to nineteen solvers and three model plug-ins.

We illustrated how **ROI** can be used to solve OPs from many different problem classes. Furthermore, we have shown how **ROI** can be used to solve challenging statistical problems like best subset selection, convex clustering and glasso. The plug-in package **ROI.plugin.msbinlp** (Hornik, Meyer, and Schwendinger 2017b) serves as an example to highlight the benefit of the development of new packages based on **ROI**. The main benefit is that there is no need to have a dependency on a specific solver which could be also interesting for the implementation of nonlinear optimization algorithms (e.g., sequential quadratic programming). Another benefit is that package authors can reuse test cases from other packages based on **ROI** and the plug-in package **ROI.tests** provides a standardized way to test new solver plug-ins.

Although **ROI** already is able to cope with a wide range of optimization prob-

lems, there are still many possibilities for extensions. These include extending the supported functional forms of the objective functions and constraints as well as adding additional solvers or model collections through plug-ins. The following gives an overview of the planned extensions.

- Currently, **ROI** is lacking a plug-in capable of solving general nonlinear optimization problems with mixed integer constraints. Within the COIN-OR project the **Couenne** (Belotti, Lee, Liberti, Margot, and Wächter 2009) and the **Bonmin** (Bonami and Lee 2013) solvers are designed to try to obtain a global solution for non-convex MINLPs. Therefore both solvers would be a valuable extension of **ROI**.

- Another popular solver from the COIN-OR project is **Ipopt** (Wächter and Biegler 2006), which aims to solve general nonlinear optimization problems. As mentioned before, there exists already the **ipoptr** package, but since there are many steps needed for the installation we assume it is currently not accessible to many R users.

- Add reader for the QPLIB file format (Furini, Traversi, Belotti, Frangioni, Gleixner, Gould, Liberti, Lodi, Misener, Mittelmann, Sahinidis, Vigerske, and Wiegele 2017).

- Add plotting methods.

- Explore possibilities of supervised solver recommendation.

Chapter 3

How to Derive Consensus Among Various Marketing Journal Rankings?

The paper was published in the May 2014 issue of the *Journal of Business Research*: Theußl S, Reutterer T, Hornik K (2014). "How to Derive Consensus Among Various Marketing Journal Rankings?" *Journal of Business Research*, **67**(5), 998–1006. doi:10.1016/j.jbusres.2013.08.006.

3.1 Introduction

Academics in the field of business research and related sub-disciplines encounter an increasing number of journal rankings (see, e.g., *Anne-Wil Harzing* via the web site http://www.harzing.com/jql.htm, for an up-to-date compilation). The majority of these journal rankings are based on stated preferences (i.e., judgments among academic peers; e.g., Fry, Walters, and Scheurmann 1985; Hult, Neese, and Bashaw 1997; Theoharakis and Hirst 2002; Schrader and Hennig-Thurau 2009), revealed preferences (i.e., citation rates as a surrogate measure of publication impact; e.g., Bakir, Vitell, and Rose 2000; Baumgartner and Pieters 2003), or a combination of the two (e.g., Azar and Brock 2008; Dubois and Reeb 2000; Zhou, Ma, and Turban 2001). While the various ranking approaches are typically fairly consistent in ranking the top tier journals in the investigated (sub-) discipline, they tend to diverge substantially as one proceeds further down the ratings. Besides the kind of data used to rank journals, the individual rankings for lower tier journals are also affected by the type of institution (e.g., academic rigor vs. practitioner orientation, type of methodological research orientation, etc.) and the geographical perspective of different groups of academics (Pieters, Baumgartner, Vermunt, and Bijmolt 1999; Tellis, Chandy, and Ackerman 1999; Theoharakis and Hirst 2002; Polonsky and Whitelaw 2005). In a comparative study of eleven rankings Mingers and Harzing (2007) found pairwise rank correlations between .32 and .79, which clearly suggests that some degree of consensus exists, but by no means complete agreement.

Despite an ongoing controversial discussion on their specific merits and drawbacks (see Polonsky 2008, for a summary), journal rankings have undoubtedly gained momentum in the evaluation of individual scholars' or academic units' research quality. For instance, the UK's higher education funding councils have adopted journal rankings as their central metrics for evaluating the research quality of universities in their regularly undertaken *Research Assessment Exercise (RAE)*(succeeded by the *Research Excellence Framework* in 2014). The widespread acceptance of the journal ranking published by the German Academic Association for Business Research (*VHB-JOURQUAL*) for assessing the research performance of business scholars in German-speaking countries is just one further example (Schrader and Hennig-Thurau 2009).

As a natural consequence of the still increasing number of available journal rankings and their growing importance to the academic community, some authors have attempted to merge compilations of single rankings into meta-rankings for various sub-disciplines such as international management (Dubois and Reeb 2000), management information systems (Rainer and Miller 2005), and innovation management and entrepreneurship (Franke and Schreier 2008). The approaches to aggregation

range from simple rank averaging (e.g., in Dubois and Reeb 2000) to more sophisticated statistical approaches such as the maximum likelihood procedure proposed by Bancroft, Gopinath, Kovács, and Rejtö (1999). The potential benefits of such meta-rankings are obvious: Firstly, they tend to balance out any biases in the single journal rankings due to raters' strategic answering behavior (often encountered with survey-based rankings) or unsystematic errors such as volatilities in the impact factors (in the case of citation-based approaches). Secondly, they provide a better guide to academics in globalized publication and job markets because they reflect an aggregate perspective of the academic community on the quality of publications (Rainer and Miller 2005; Franke and Schreier 2008).

However, two major challenges are inherent in aggregating journal ranking data sets. At the time of this writing neither of which has been adequately resolved by the existing approaches: (1) the different measurement scales used by the rankings and (2) incomplete information (Franke and Schreier 2008; Mingers and Harzing 2007; Schrader and Hennig-Thurau 2009). The first issue refers to the fact that individual rankings make use of quite different scales, including binary (yes/no), ordinal (e.g., grades A+, A, B, etc.), and numeric scores (e.g., impact factors) to construct their rankings. This issue makes the aggregation of rankings using conventional statistical methods cumbersome. The second issue, which is related to the typically large number of "missing observations" in ranking data sets, appears to be even more problematic. Such missing information accrues because the various rankings cover only subsets of journals, which usually coincide only partially. Thus, the sparsity of data sets generally increases when including a broader set of rankings.

In the paper, we present a method for deriving meta-rankings of journals by solving consensus optimization problems. The proposed methodology obtains consensus rankings from paired comparisons among a set of individual rankings, can accommodate mixed types of measurement scales and is relatively robust even when used for sparse data. The next section addresses the challenges involved in suitably aggregating rankings to derive a consensus ranking, outlines the cornerstones of the proposed consensus ranking methodology, and provides pointers for the computational implemention of the related procedures. The paper proceeds by introducing the journal ranking data that the study uses. The following section shows the results of applying the method, using a subset of marketing-related journals, which the *Harzing Journal Ranking Repository* provided. The presentation and discussion of our results show that even though the investigated rankings of marketing-related journals are far from being identical, the proposed method can produce a consensus ranking which shows a considerably high level of agreement with the individual rankings. We also explore the sensitivity of the consensus ranking methodology to variations in the number of rankings and/or journals, by studying stability and de-

generation issues. In addition, we include a comparison of the consensus ranking methodology to a naive ranking, derived through a simple averaging of ranks, which demonstrates the benefits of the consensus ranking method. Finally, we conclude with a discussion on the value of consensus rankings and by outlining some points for future research.

3.2 Methodology

As a starting point of the problem of finding adequate meta-rankings of various journal rankings, consider a total of B journal rankings $\mathcal{R} = \{R_1, \ldots, R_B\}$ and n journals $\mathcal{J} = \{J_1, \ldots, J_n\}$. Table 3.1 shows an example ranking data set for $B = 4$ and $n = 3$ extracted from the Harzing Journal Ranking Repository for illustration purposes.

	R_1	R_2	R_3	R_4
J_1	A+	D	4	33.50
J_2	A	B	3	
J_3	A	B	3	23.30

Table 3.1: Example ranking data set.

Three of the four rankings use an ordinal scale, with levels A+, A, B, C, D (rankings R_1 and R_2), and scores of four to one, respectively (ranking R_3; ordered from highest to lowest as defined in the ranking description). Ranking R_4 uses a metric scale of 0 to 100, where a higher value indicates better journal quality.

In the days of the French revolution, Borda (1781) (who computed the sum of individual rankings) and Condorcet (1785) (who used the majorities of paired comparisons) proposed the first heuristic aggregation rules for constructing meta-rankings; see for example Cook (2006) for a survey of consensus models of ordinal preference rankings.

Leaving ranking R_4 aside so as to avoid dealing with the missing observation for J_2, and applying the Borda aggregation rule, gives an aggregate evaluation that all three journals are of the same quality, since the sum of ranks is 5 in all cases ($1 + 3 + 1$ for J_1 and $2 + 1 + 2$ for J_2 and J_3).

As ordinal rankings are typically not Likert-type (interval) scaled and may have different granularities, treating grades like A+, A as rank scores (i.e., 1, 2, etc.) is quite problematic, and the above aggregation rule may lead to results which do not necessarily reflect the quality assessments behind the "majority" of rankings. In the example above, at least two rankings put J_1 above J_2 and J_3 and only one ranks J_1 below the other two journals. However, the consensus shows indifference between all three journals.

An approach based on pairwise comparisons of journals appears to be more adequate. Comparing journals within each applicable ranking by investigating whether a ranking evaluates a journal J_i as explicitly superior to another journal J_j implies the relationship $J_i > J_j$. The results of these paired comparisons for a particular journal ranking R_b can be represented by an incidence matrix $I_b = [\chi_{ij}(R_b)]$, where the entries are the incidences $\chi_{ij}(R_b) = 1$ if $J_i \leq J_j$ or zero otherwise. This encoding warrants that the existence of ties (i.e., equivalent preferences) is taken into account. Using such an approach we do not necessarily need to interpret all available data as metric (such as e.g., Mingers and Harzing 2007). Thus, the direct processing of ordinal data that journal ranking publishing institutions commonly provide becomes feasible. Furthermore, if journal J_i is not included in ranking R_b, then necessarily $\chi_{ij}(R_b) = \chi_{ji}(R_b) = 0$ for all $j \neq i$. This way of encoding only manifest "preference" allows us to deal with missing paired comparison data. The incidence matrices corresponding with the rankings from Table 3.1 are shown in Table 3.2.

	I_1			I_2			I_3			I_4		
	J_1	J_2	J_3	J_1	J_2	J_3	J_1	J_2	J_3	J_1	J_2	J_3
J_1	1	0	0	1	1	1	1	0	0	1	0	0
J_2	1	1	1	0	1	1	1	1	1	0	0	0
J_3	1	1	1	0	1	1	1	1	1	1	0	1

Table 3.2: Incidence matrices for rankings R_1, R_2, R_3, and R_4. A non-zero entry denotes a $\leq$ relationship between two journals.

The incidence matrix I_1 for ranking R_1 encodes the results of the following paired comparison: $J_1 \leq J_1$, $J_1 > J_2$, $J_1 > J_3$; neither J_2 nor J_3 is ranked higher than the other, that is $J_2 \sim J_3$ (where $\sim$ denotes indifference between two journals). In contrast, the incidences for R_2 imply that $J_2 > J_1$ and $J_3 > J_1$. For R_3 the incidences are identical to the ones derived from R_1. I_4 shows that $J_1 > J_3$ but no other comparisons exist.

By encoding the paired comparisons in this way, we can employ the following geometric idea: if one can measure the distance between rankings, then the "center" of the given collection of rankings, which minimizes the total distance to all rankings, is a natural and intuitively appealing choice for consensus (see Régnier 1965). One can derive the distance between two rankings by counting the number of discordant entries in the corresponding incidence matrices (i.e., the number of discordant paired comparison results). Formally, the distance $d(R_k, R_l)$ between R_k and R_l is equal to

$$\sum_{i,j} |\chi_{ij}(R_k) - \chi_{ij}(R_l)|.$$

For example, the distance between R_1 and R_2 is 4. Following this conceptual

understanding, the complete ranking R, which minimizes $\sum_b d(R_b, R)$, is the consensus ranking. In the above example, out of all possible R, the consensus ranking is $J_1 > J_2 \sim J_3$. This ranking is equivalent to the encoded incidence matrices I_1 and I_3. Here no better solution exists; the aggregate distance between R and all of the other rankings involved is minimal.

Mathematically, complete rankings are "preference relations", or *weak orders*, defined as binary relations on a set of objects which are complete, reflexive and transitive (see Fishburn 1972, for a complete reference). Note that such weak orders permit ties (i.e., they do not enforce strict preference, or asymmetry). For larger problems explicitely going through all possible incidence matrices in order to find the one minimizing the aggregate distance may require rather too much effort. Following Hornik and Meyer (2007), one can obtain the consensus incidence matrix by solving the following binary program

$$\sum_{i \neq j} c_{ij} \chi_{ij}(R) \Rightarrow \max$$

where $c_{ij} = \sum_b (2 w_b \chi_{ij}(R_b) - 1)$ and the $\chi_{ij}(R)$ must be constrained such that $R \in \mathcal{C}$, the set of consensus candidates. In the case of complete rankings, the constraints defining $\mathcal{C}$ are:

$$\begin{array}{rcll} r_{ij} & \in & \{0,1\} \quad i \neq j & \text{(binarity)} \\ r_{ii} & = & 1 & \text{(reflexivity)} \\ r_{ij} + r_{ji} & = & 1 \qquad i \neq j & \text{(completeness)} \\ r_{ij} + r_{jk} - r_{ik} & \leq & 1 \qquad i \neq j \neq k & \text{(transitivity)} \end{array}$$

(note that the constraints in Hornik and Meyer 2007, are not for rankings). Any state-of-the-art mixed integer programming solver can solve this combinatorial optimization problem known to be computationally complex ($\mathcal{NP}$-complete, see Wakabayashi 1998). Mathematically speaking, the above binary program determines the consensus of the (possibly incomplete) rankings $R_1, \ldots, R_B$ as the complete ranking R, which minimizes the total dissimilarity between itself and the original rankings:

$$\sum_{b=1}^{B} d(R, R_b) \Rightarrow \min_{R \in \mathcal{C}},$$

For complete rankings, the above illustrated dissimilarity measure d is the "symmetric difference" distance, which Kemeny and Snell (1962) shows to be the "natural" distance between complete rankings, in the sense of uniquely satisfying a set of basic axiomatic conditions. Note that, for a given collection of rankings, a unique solution to the above binary optimization problem does not necessarily exist. How-

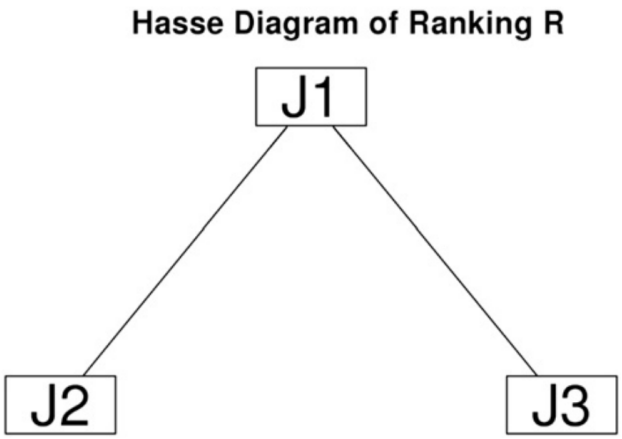

Figure 3.1: Consensus journal ranking for the journals J_1, J_2, and J_3.

ever, using branch and cut approaches, one can identify all consensus solutions and use the commonalities in these solutions to obtain a robust understanding of the underlying preference structure.

For the applications in the paper, we make use of the R system (version 3.6.2) for statistical computing (R Core Team 2019a) to carry out all computations, and generate the problem definition, based on the supplied data set, using functions and methods available in the **relations** package (Hornik and Meyer 2010). Furthermore, we use the R package **Rcplex** (Theußl and Bravo 2016), which provides an interface to the commercial optimizer **CPLEX** (ILOG 2019), to solve all optimization problems.

From the resulting consensus ranking R, we can compute a numerical rank for each object, so-called preference scores or "generalized ranks". Preference scores are a linear transformation of the difference between lost and won paired comparisons into the range from 1 to the number of objects n (see Regenwetter and Rykhlevskaia 2004, for details). For the application at hand, the unique solution to the corresponding optimization problem is $J_1 = 3 > J_2 \sim J_3 = 2$. Thus, I_1 and I_3 indeed represent the incidence matrix of R. Consensus rankings typically include many ties between journals (i.e., groups of journals with equal scores) and thus so-called Hasse diagrams can represent them nicely. Such diagrams denote a class of data representation techniques, which are capable to visualize weak-ordered relations among objects among objects (available as a plot method in the R package **relations**; cf., e.g., Freese 2004). Figure 3.1 shows such a Hasse diagram for the above consensus ranking.

3.3 Data set characteristics

To illustrate the performance of the consensus ranking approach, we apply the above described methodology to a set of 12 renowned rankings of journals, related to the marketing discipline. They are all available from published sources on the Web and included in the 34^{th} edition of the Harzing Journal Quality List (https://www.harzing.com). Table 3.3 provides a complete list of the journal rankings used, showing publishing institution and the corresponding abbreviation. Confronted with all 851 journals available in this data set, we asked domain experts

Abbreviation	Institution
ABDC08	Australian Business Deans Council
ABS09	Association of Business Schools
Ast08	Aston Business School
Bjm04	Business and Management 2001 RAE/UK
Cnrs08	Comite National de la Recherche Scientifique
Cra09	Cranfield University School of Management
EJL06	Erasmus Research Institute of Management
Hkb05	Hong Kong Baptist University School of Business
Theo05	ALBA Journal Ranking
UQ07	University of Queensland
Vhb08	Verband der Hochschullehrer fuer Betriebswirtschaft
Wie01	WU Vienna University of Economics and Business

Table 3.3: Journal rankings and corresponding abbreviations.

to select those journals which they considered to be potential publication outlets for marketing academics' research. Following a similar procedure to that of Dubois and Reeb (2000), the experts then had to assign each of the selected journals to one of the following two categories: (1) A *core* list of marketing journals with an inherent focus on general or specific topics in marketing. (2) An *extended* list including journals from adjacent disciplines, but which also have marketing academics as their target audience. The latter list includes journals focusing on disciplines such as General Business Research, Information Science, Applied Psychology, and Operations Research.

To obtain robust interpretable consensus solutions from the integer optimization problem presented in the last section, we removed those journals which a significant majority (three quarters) of the rankings does not rank. This selection procedure resulted in a final sample of 33 journals in the core list and 62 journals in the extended list (which includes the core list). Table 3.4 lists all of these journals and their abbreviations; those in the core list are marked with a C.

Notice that the *Journal of Business Research* is in the core list because a substantial portion of that journal's published articles are related to the marketing

Journal Name	Abbrev	Journal Name	Abbrev
Advances in Consumer Research (C)	ACR	Journal of Interactive Marketing (C)	JIntMar
California Management Review	CMR	Journal of International Marketing (C)	JIM
Computers & Operations Research	COR	Journal of Macromarketing (C)	JMK
Decision Sciences	DS	Journal of Marketing (C)	JM
Decision Support Systems	DSS	Journal of Marketing Management (C)	JMM
European Journal of Information Systems	EJIS	Journal of Marketing Research (C)	JMR
European Journal of Marketing (C)	EJM	Journal of Personal Selling and Sales Management (C)	JPSS
European Journal of Operational Research	EJOR	Journal of Product Innovation Management (C)	JPIM
European Management Journal	EMJ	Journal of Public Policy & Marketing (C)	JPPM
Harvard Business Review	HBR	Journal of Retailing and Consumer Services (C)	JRCS
Industrial Marketing Management (C)	IMM	Journal of Retailing (C)	JR
Interfaces	Int	Journal of Service Research (C)	JSR
Int. Business Review	IBR	Journal of Services Marketing (C)	JS
Int. Journal of Advertising (C)	IJA	Journal of Small Business Management	JSBM
Int. Journal of Electronic Commerce	IJEC	Journal of Strategic Marketing (C)	JSM
Int. Journal of Logistics Management	IJLM	Journal of the Academy of Marketing Science (C)	JAMS
Int. Journal of Market Research (C)	IJMR	Journal of the Operational Research Society	JORS
Int. Journal of Research in Marketing (C)	IJRM	Journal of World Business	JWB
Int. Journal of Retail & Distrib. Man. (C)	IJRDM	Marketing Letters (C)	ML
Int. Journal of Service Industry Man. (C)	IJSIM	Marketing Science (C)	MkS
Int. Marketing Review (C)	IMR	MIS Quarterly	MISQ
Journal of Advertising (C)	JA	Psychology and Marketing (C)	PM
Journal of Advertising Research (C)	JAR	R&D Management	RM
Journal of Applied Psychology	JAP	Research Policy	RP
Journal of Business Ethics	JBE	Service Industries Journal	SIJ
Journal of Business Research (C)	JBR	Sloan Management Review	SMR
Journal of Business Venturing	JBV	Small Business Economics	SBE
Journal of Consumer Marketing (C)	JCM	Strategic Management Journal	SMJ
Journal of Consumer Psychology (C)	JCP	Supply Chain Management: An International Journal	SCMAIJ
Journal of Consumer Research (C)	JCR	Thunderbird International Business Review	TIBR
Journal of Forecasting	JOF	Total Quality Management & Business Excellence	TQMBE

Table 3.4: Journals used in this study (C indicates membership in the *core* list).

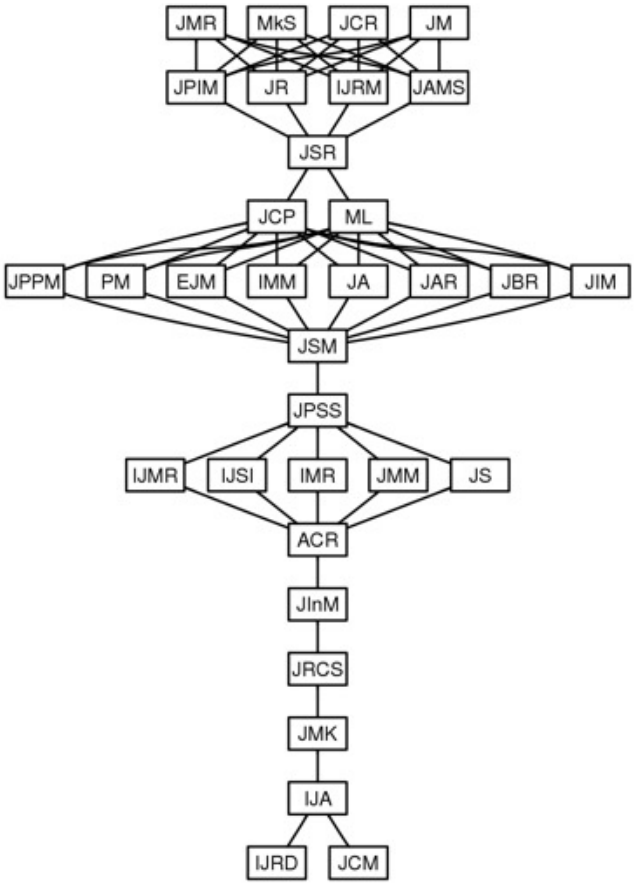

Figure 3.2: Consensus journal ranking for the *core* journal list.

discipline. The ranking data set of the journals in Table 3.4, including descriptions and the rankings employed, is available in an online appendix, which we prepared for the paper, at https://statmath.wu.ac.at/projects/jcr/.

3.4 Consensus ranking results

In this section we present, visualize, and discuss the results of the proposed method. To illustrate the effect on the consensus ranking of adding journals from adjacent disciplines to the core list of marketing journals, we perform separate analyses for the core and the extended journal lists. Furthermore, we investigate the sensitivity of the consensus solution to variations in the number of journals and rankings used, study potential degeneration effects, and compare the similarity/dissimilarities of the consensus to a naive meta-ranking as well as to individual rankings.

3.4.1 Core versus extended list

Figure 3.2 portrays the Hasse diagram of the consensus ranking for the core journal list. The diagram clearly illustrates that, in the consensus across the 12 rankings,

the marketing journals are arranged in several tiers. Each tier indicates the same degree of preference for the journals in that tier. For example, and indeed not very surprisingly, the top-tier marketing journals are the *Journal of Consumer Research* (JCR), the *Journal of Marketing* (JM), the *Journal of Marketing Research* (JMR), and *Marketing Science* (MkS). The second tier of the consensus preference structure also contains high-quality journals, such as the *International Journal of Research in Marketing* (IJRM), the *Journal of the Academy of Marketing Science* (JAMS), the *Journal of Product Innovation Management* (JPIM), and the *Journal of Retailing* (JR). In this respect our findings of this research are consistent with the conclusions of many other authors that a high degree of agreement among academics as to which are the top journals in their discipline seems to exist (Polonsky and Whitelaw 2005; Theoharakis and Hirst 2002).

Interestingly, this consensus preference structure suggests that the *Journal of Service Research* (JSR) separates the abovementioned top-level marketing journals from lower-level publication outlets. As one proceeds further down the ladder, more specialized, niche marketing journals "join the crowd". Quite obviously, the derived consensus preference structure tends to distinguish between two broad types of marketing journals: The first, in the top levels, with a relatively broad scope, covering a wide range of topics, and the second, with more focused positioning, in the lower levels of the ranking list.

Adding in journals from adjacent disciplines by using the extended journal list, we observe a unique structure of consensus rankings for the top journals (see the Hasse diagram shown in Figure 3.3). Compared to the core list of journals, the *Journal of Applied Psychology* (JAP), *Management Science* (MS), *MIS Quarterly* (MISQ), *Operations Research* (OR), and the *Strategic Management Journal* (SMJ) join the top tier of marketing-related journals. The inclusion of JAP in this group might be slightly surprising, but this journal is extremely highly ranked in the journal quality rankings published by UK and Australian institutions. The same marketing journals from the core list remain in the second tier. Now, however, some journals with a relatively broad disciplinary scope but with a focus on quantitative research methodologies (EJOR, DS) and information science (DSS, EJIS) join the group. Also, the rankings deem the most prestigious practitioner-oriented transfer journals in the management discipline (HBR, SMR, CMR) to be highly-esteemed publication outlets for marketing academics.

The move of ML (which the UK- and continental-European-based journal rankings involved in our study evaluate comparatively high) from a lower tier to the second tier, and the inversion of JCP and JSR in the consensus ranking for the extended list are two of the most noticeable differences in the preference structure which emerges from extending the journal list in the paper's proposed optimization

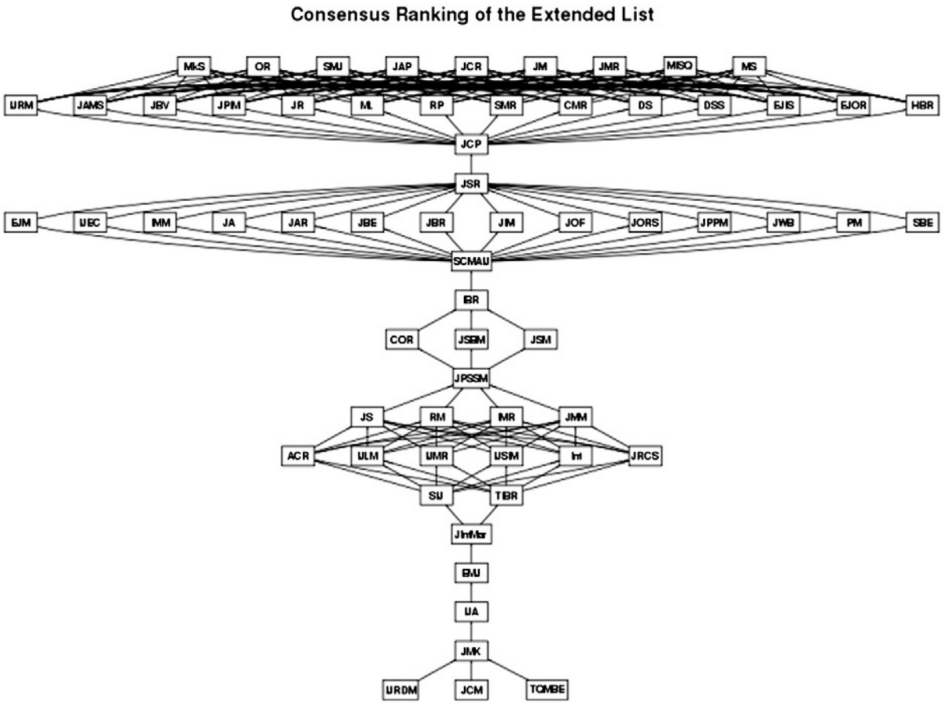

Figure 3.3: Consensus journal ranking for the *extended* journal list.

approach. In fact, the consensus preference ranks for these three journals are in roughly the same range, and are at a greater distance from the higher-ranked group of journals (top tier) and those in lower levels. Again, as already noted for the consensus ranking using the core journal list, the journals ML, JCP, and JSR are responsible for dividing the complete group of marketing-related journals into these two broader subsets.

3.4.2 Stability of consensus solutions and degeneration effects

The above discussion of the properties of the two consensus solutions depicted in Figures 3.2 and 3.3 shows that the number of ties apparently increases substantially when we add more journals to the core list of marketing journals. In the following, we provide the results of a systematic sensitivity analysis of consensus ranking solutions, involving the use of different (subsets of) journals and rankings. Of particular interest are two questions: (1) Is the consensus solution robust to the inclusion of different journals and rankings? (2) Does the consensus solution show a tendency towards indifference (i.e., does the solution rank many journals equally) when one adds further journals and/or rankings? While the first question addresses the stability of consensus solutions, the second investigates potential degeneration effects in the derived solutions.

In order to investigate these two properties of consensus solutions, we conduct

a bootstrap experiment. Our bootstrapping approach is defined as follows: we randomly draw a given number q journals from $\mathcal{J}$ and m rankings from $\mathcal{R}$, where $q \in \{10, 20, \ldots, 60\}$ and $m \in \{2, 3, \ldots, 12\}$, and compute the consensus ranking for the corresponding collection. We then repeat this procedure 1,000 times for each possible combination of q and m. To measure the stability of the consensus solutions we compute the Kendall rank correlation coefficient (Kendall's τ) for the result of each bootstrap sample against a suitably matched subset from the consensus ranking, derived for all of the rankings and journals included in the extended list.

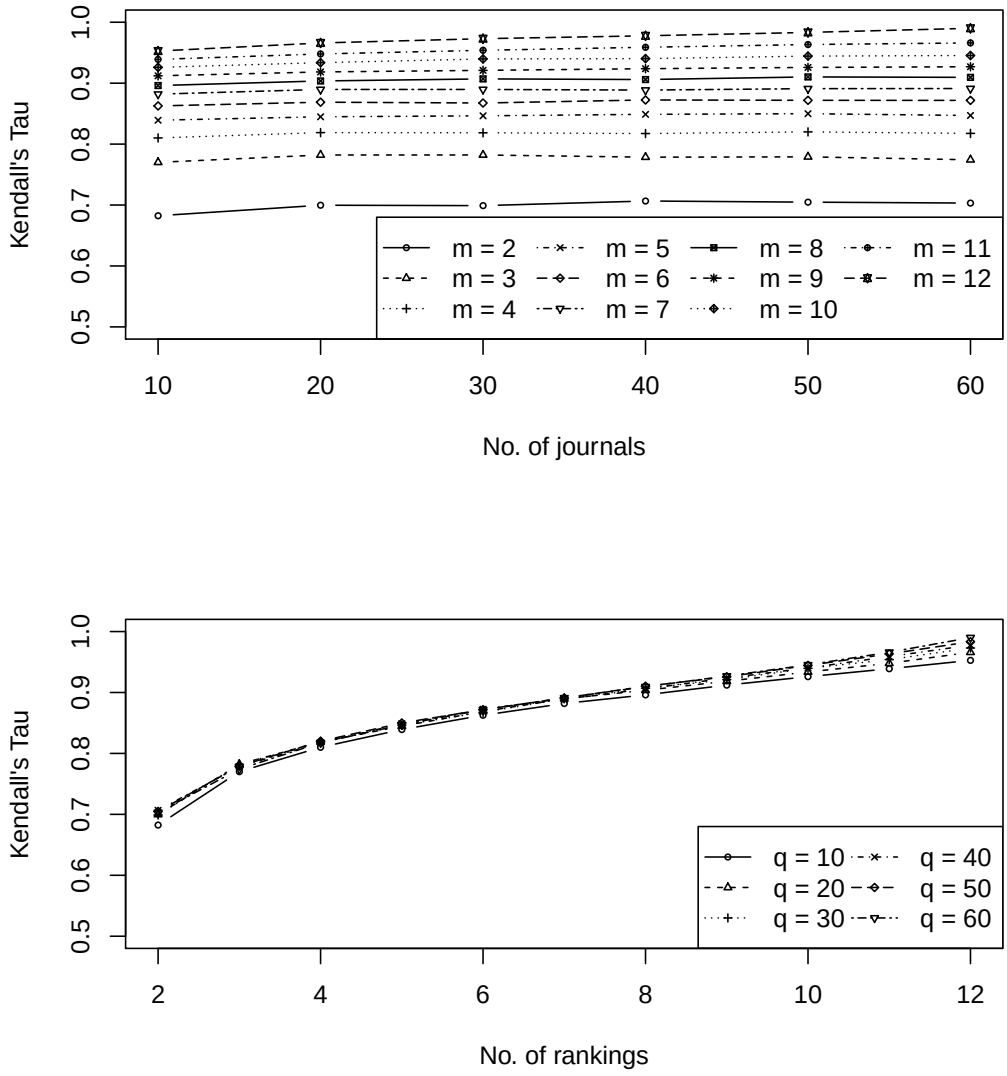

Figure 3.4: Development of average rank correlations between the overall and bootstrap sample consensus rankings.

Figure 3.4 plots the development of the average rank correlations between the overall consensus ranking and those derived for the bootstrap samples, for increasing

numbers of involved journals q and rankings m, respectively. The plots clearly show a much more marked dependence of the rank correlations on the number of involved rankings than on the number of journals. While, for a given number of rankings, the inclusion of additional journals does not affect the stability of the derived solution much, the lower part of Figure 3.4 shows that, when the number of rankings m increases, the rank correlation coefficient (almost linearly) converges to almost 1. This result is not very surprising, because one would expect to have much more difficulty finding an agreement between raters when considering an extra rater's opinion about a given set of journals, than in the case where a given set of raters has to rank a new journal. Regressing the average rank correlations on the two treatment factors q and m of our bootstrap experiment, as well as their interactions (the saturated model), confirms the visual diagnostic statistically. Using backward selection, one can reduce the complexity of the model, so that m remains as the only significant parameter, given a significance level of 0.05. The parameter estimate for the number of rankings m is 0.02 and explains more than 92% of the variance of the average rank correlation ($r^2 = 0.92$; with a p-value for the model of $p < 0.01$).

To investigate potential degeneration effects for the derived consensus rankings, depending on variations across the number of available journals q and rankings m, we compute the average number of ties per involved journal, in the consensus ranking, across all bootstrap replications. In contrast to the findings regarding the stability of consensus rankings for subsets of available journals and rankings, this test shows the opposite effect.

Figure 3.5 depicts results which show that employing more journals to construct the consensus ranking increases the number of ties, regardless of the number of rankings one uses. This outcome seems quite reasonable, because the more journal quality rating opinions from different raters one must aggregate, the higher the tendency towards indifference one would expect. One can also observe this property of consensus solutions when comparing Figure 3.2 against Figure 3.3. Using the same backward selection procedure as above, we find that only the number of journals q is statistically significant in explaining the suggested degeneration measure. The parameter estimate for q is 0.13 and explains more than 87% of the variance of the degeneration measure ($r^2 = 0.88$; $p < 0.01$).

3.4.3 Comparison of the consensus with naive and individual rankings

In view of the insights gained so far, an obvious question that arises is how dissimilar the individual rankings are from the derived consensus rankings. Furthermore, how the proposed consensus ranking methodology performs against simpler methods of

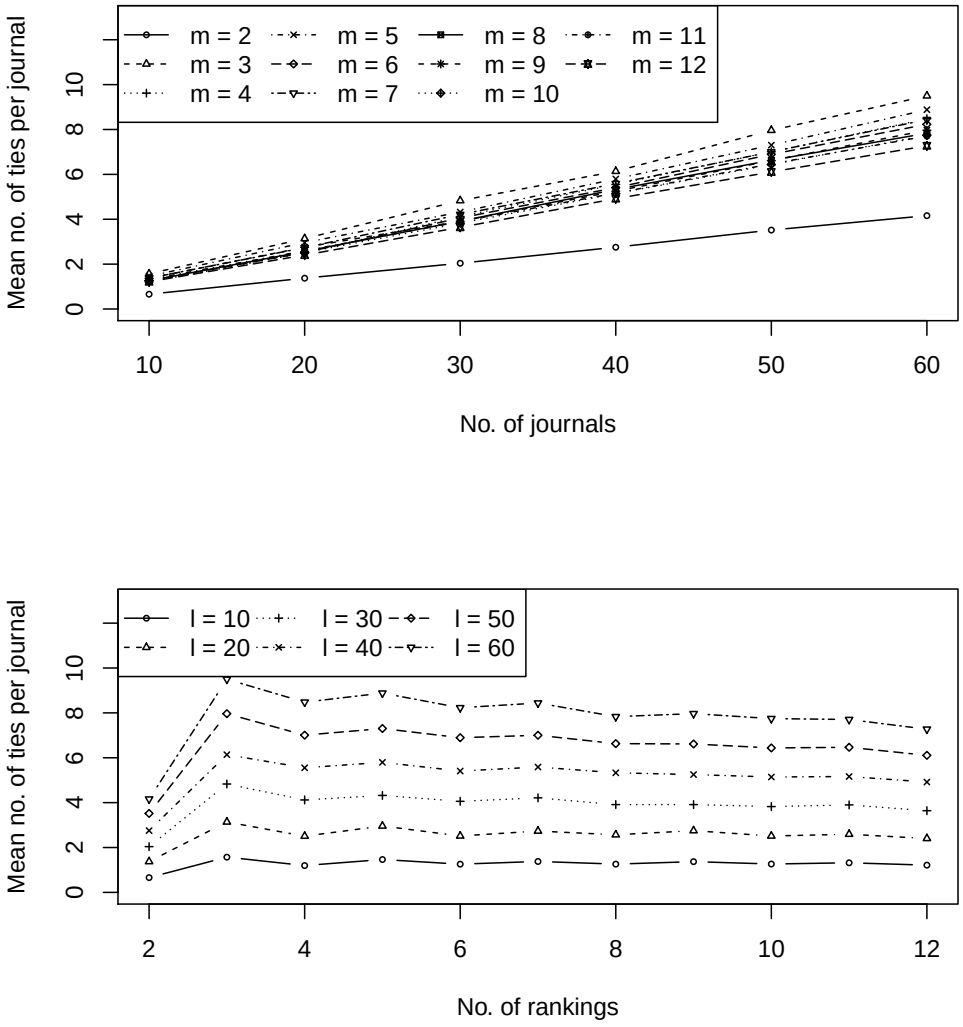

Figure 3.5: Development of the average number of ties per journal over all bootstrap replications.

aggregating individual rankings is of interest. In an attempt to investigate this issue in more detail, we employ an averaging procedure similar to that used by Dubois and Reeb (2000). To derive such a naive meta-ranking, we compute rescaled preference scores, for each individual ranking for each journal, and sum them. Then, we transform the resulting meta-score into a rank order to represent the meta-ranking.

To compare the relative performance of the ranking methods, we use the absolute sum of Kemeny-Snell distances d derived for each of the available rankings (including the consensus and the naive meta-ranking) vis-a-vis all other rankings, as well as the corresponding average rank correlation coefficient (Kendall's τ), as critera for the representation quality of the meta-rankings. Table 3.5 shows them, together with

the identifiers of the respective rankings, in descending order of d. The smallest

	d	Kendall's τ
Consensus	12140	0.61
Naive	12904	0.59
ABDC08	13930	0.58
ABS09	14276	0.58
Ast08	14814	0.55
UQ07	15324	0.55
Vhb08	15930	0.49
Cra09	17110	0.51
EJL06	18410	0.45
Cnrs08	18992	0.45
Bjm04	19380	0.38
Wie01	20228	0.37
Hkb05	20444	0.38
Theo05	21442	0.42

Table 3.5: Absolute Kemeny-Snell distances and average rank correlations for each ranking in comparison to all others.

distance is between the consensus ranking and all other rankings. Of course, this finding is not surprising, because the consensus ranking by definition represents the global minimum for the given optimization criterion (which is based on the Kemeny-Snell distance). However, this ranking also has the highest average rank correlation, which reflects its ability to serve as a superior meta-ranking. According to these measures, the naive ranking is the second-closest to all other rankings. At first sight, the ranking appears quite similar to the consensus ranking in terms of ordering (at least for the top-ranked journals). For example, the seven most preferred journals according to the consensus ranking are also the top seven journals according to the naive meta-ranking.

However, a more detailed examination of the actual positions the journals occupy in the aggregated rankings suggests that the consensus ranking method is a more adequate meta-ranking representation than the naive method. For an illustration, consider the top 30 journals according to the naive meta-ranking; Table 3.6 lists these journals, along with their respective preference scores and corresponding positions in the consensus solution. While the JM appears in the group of the highest ranked journals in the consensus ranking, the JM is seventh according to the naive meta-ranking, even though the journals's naive meta-scores are relatively close to those of SMJ and MISQ. The problem with this naive kind of aggregation becomes even more apparent if one compares the performance of JM in the individual rankings to that of journals ranked higher according to the naive meta-ranking. In such a comparison, JM wins more pairwise comparisons against SMJ and MISQ than the other way round. More substantial differences occur if one proceeds further down

	NR: scores	CR: ranks
MkS	-18.96	1
JCR	-17.50	1
JMR	-16.63	1
SMJ	-15.96	1
MISQ	-15.72	1
JM	-15.59	1
JAP	-14.80	1
JAMS	-12.48	2
RP	-10.70	2
DS	-10.41	2
JR	-10.27	2
JPIM	-9.41	2
IJRM	-8.64	2
JBV	-7.12	2
DSS	-6.58	2
HBR	-6.30	2
ML	-6.13	2
EJIS	-5.47	2
CMR	-4.91	2
EJOR	-4.35	2
SMR	-4.19	2
JCP	-2.71	3
SBE	-0.92	5
JBR	-0.89	5
JSR	-0.64	4
JWB	-0.27	5
IJEC	-0.22	5
JORS	-0.18	5
JOF	-0.00	5
JIM	0.13	5

Table 3.6: Preference scores for the top 30 journals from the naive meta-ranking (NR) compared to the consensus ranking (CR).

the ranking (for a more detailed inspection, see the online appendix of the paper at https://statmath.wu.ac.at/projects/jcr/).

From a theoretical point of view the discussion about the performance of the naive aggregation method goes back to the dispute between Borda (1781) and Condorcet (1785) and has produced a variety of arguments indicating that this method is often unsuccessful in representing the "true" majority decision. From a more practical perspective, another argument exists which makes the properties of a consensus ranking appealing: the permission of ties in the consensus ranking provides an evident and intuitively comprehensible basis for deciding on the appropriate cut-off points that discriminate between journals of several quality tiers (e.g., deciding on a boundary between A- and B-journals). In the case of naïve aggregation, this task becomes cumbersome and is much more dependent on discretionary human

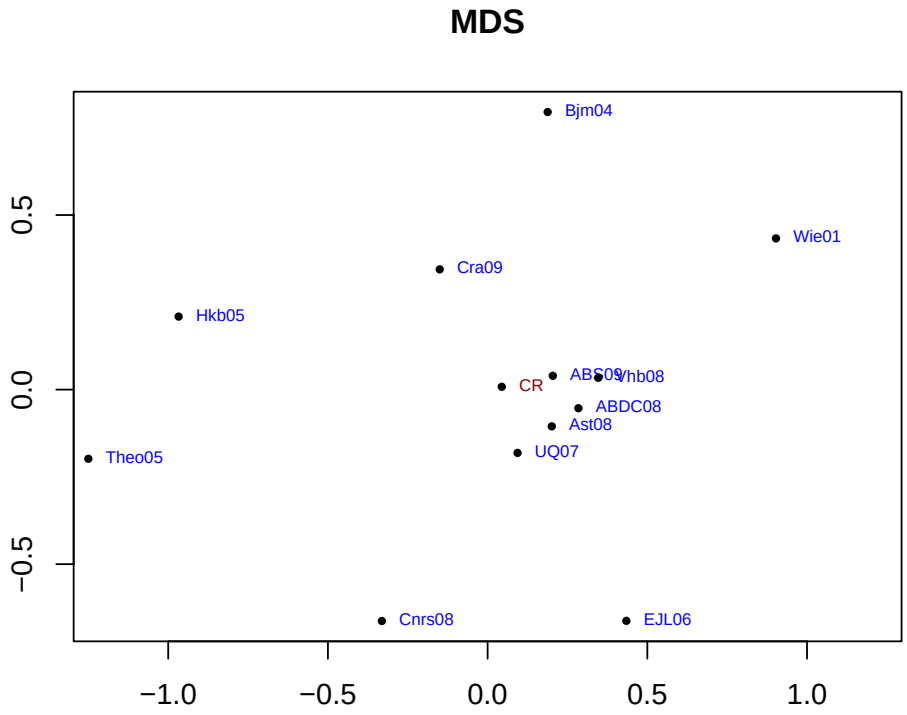

Figure 3.6: Symmetric difference distance-based representation of the individual rankings and the consensus ranking.

judgments.

To illustrate the consensus ranking's goodness of representation quality for the underlying individual rankings, Figure 3.6 portrays the pairwise distances between the involved journal rankings, d, in a two-dimensional display. The configuration results from using the multidimensional scaling (MDS) technique SMACOF (de Leeuw and Mair 2009; Mair, de Leeuw, and Groenen 2019), which minimizes a stress measure by means of majorization (the nonmetric stress for the solution is 0.121).

The consensus ranking CR clearly appears in the center of the two-dimensional plot. In general, the further a rankings's position is from the consensus, the lower the level of agreement. Quite obviously, the individual rankings concentrated around the consensus ranking share some potential to adequately reflect the aggregate perspective of the academic community in evaluating the quality of marketing journals, while those rankings that are further away fail to do so. In this respect, distance from the consensus ranking could also serve as an indication of the capability of an individual ranking to fulfill the properties of an appropriate global meta-ranking.

3.5 Conclusions

Journal rankings have become an important tool to assess the research quality of publications. Academics show widespread consent that focusing on a single journal ranking is risky and inadequate at reflecting the aggregate perspective of the academic community as to the quality of research publication outlets. Prior attempts to merge compilations of single rankings into suitable meta-rankings struggle with the different measurement scales used by the various rankings and the issue of incomplete information. The paper presents an optimization-based approach, demonstrating how one may derive consensus rankings from several individual ones. The approach is capable of accounting for different scale levels (numeric, ordinal) and partial intersections of the journal sets included in the aggregation.

We apply the proposed consensus ranking method to various subsets of marketing-related journals included in the *Harzing Journal Quality List*. Even though the single rankings are rather divergent for lower-ranked journals, the results show that one can derive a consensus ranking with a considerably high level of agreements with the original set of single rankings. Notwithstanding these results, one must be careful in drawing conclusions from such an analysis, because the results depend on the journal rankings one uses. The sensitivity analysis clearly shows that the the number of individual rankings affects the stability of the derived consensus meta-ranking, whereas the solution tends to degenerate as the size of the journal list explodes.

However, the application to marketing-related journals also demonstrates the superiority of the consensus ranking over a simpler approach involving rank averaging. Compared to previous rather complicated and extensive efforts to adequately aggregate single rankings into meta-rankings, our approach is easily implementable by using ready-to-use computational resources and applicable to a wide range of similar ranking aggregation tasks. Instead of requiring sometimes incomprehensible interventions by the analyst, the proposed procedure relies on a formal solution of the underlying optimization problem and thus produces an optimum level of agreement over the derived meta-ranking, among the set of single rankings. Thus, our findings should encourage researchers and, in particular, research assessment institutions to adopt a route that allows them to objectify their ranking efforts. This approach could contribute towards avoiding much of the sometimes very emotional and controversial discussion among academics about single domain-specific rankings.

Chapter 4

A tm Plug-In for Distributed Text Mining in R

The paper was published in the November 2012 issue of the *Journal of Statistical Software*:

Theußl S, Feinerer I, Hornik K (2012). "A **tm** Plug-In for Distributed Text Mining in R." *Journal of Statistical Software*, **51**(5), 1–31. doi:10.18637/jss.v051.i05. URL http://www.jstatsoft.org/v51/i05.

4.1 Introduction

In the information age statisticians are confronted with an ever increasing amount of data stored electronically (Gantz, Chute, Manfrediz, Minton, Reinsel, Schlichting, and Toncheva 2008). This is in particular true for the most natural form of storing information, namely text. Many interesting research questions can be answered via statistical analysis or modeling of huge text corpora. For example news agencies like Reuters provide access to databases of news text documents annotated with rich semantic metadata. These can be employed, e.g., to check daily sentiment in newspaper articles in order to measure interactions between the media and the stock market (Tetlock 2007). E-print repositories such as the arXiv (`http://arXiv.org/`) or the CiteSeerX project (`http://citeseerx.ist.psu.edu/`) allow for harvesting metadata and open access to the corresponding content (i.e., download of full text articles). This information can be used for bibliometric and scientometric analyses, e.g., to find patterns like the formation or development of author or topic (Blei and Lafferty 2007; Griffiths and Steyvers 2004) networks. Furthermore, access to documents written in several centuries, such as the books made available in project Gutenberg (`http://www.gutenberg.org/`), allows one to study how linguistic patterns develop over time. In a publication in Science, Michel *et al.* (2011) use 15% of the digitized Google books content (4% of all books ever printed) to study the diffusion of regular English verbs and to probe the impact of censorship on a person's cultural influence over time. This led to the advent of a research field called *Culturomics*, the application of high-throughput data collection and analysis to the study of human culture.

R (R Core Team 2019a) has gained explicit text mining support via the **tm** package (Feinerer 2018) originally presented in Feinerer *et al.* (2008). This infrastructure package provides sophisticated methods for document handling, transformations, filters, and data export (such as constructing document-term matrices). With a focus on extensibility based on generic functions and object-oriented inheritance, **tm** makes it possible to apply a multitude of existing methods in the R world to text data structures as well.

However, the endeavor to analyze huge text corpora using **tm** is the source of two challenges: (1) the amount of data to be processed in a single machine is usually limited by the available main memory (i.e., RAM), and (2) an increase of the amount of data to be analyzed leads to higher demand for efficient procedures for calculating valuable results. Thus, it is highly imperative to find a solution which overcomes the memory limitation (e.g., by splitting the data into several pieces) and to markedly reduce the runtime by distributing the workload across available computing resources (such as CPU cores or virtual machine instances). Typically,

we consider distributed memory platforms like clusters of workstations for such applications since they are scalable in terms of CPUs and memory (disk space and RAM) employed. Furthermore, many different programming models and libraries like the message passing interface (MPI) facilitate working with this kind of *high performance computing* systems. Many of those libraries can directly be employed in R (see Schmidberger *et al.* 2009, for further references). Still, one open question remains: is there an efficient way to handle large corpora using R?

In the paper we show that by using the MapReduce distributed programming model (Dean and Ghemawat 2008), and a corresponding implementation called **Hadoop** (The Apache Software Foundation 2019), we are able to transparently distribute the documents on one or several storage entities, apply functions on the subsetted corpus possibly in parallel, and gather results on a cluster of workstations or other (distributed) computing platforms. Typical tasks in text mining like preprocessing can easily be run as parallel distributed tasks without knowing details about the underlying infrastructure. The corresponding extensions to make **tm** recognize such a distributed programming model are encapsulated in a separate plug-in package called **tm.plugin.dc** (Theußl and Feinerer 2015) offering a seamless integration building on functionality provided by interfaces to MapReduce environments.

The remainder of the paper is organized as follows. In Section 4.2 we review the typical workflow using the **tm** package and indicate challenges which need to be tackled when working with large data sets. Section 4.3 summarizes the key concepts of the MapReduce programming model and how it is usually applied in a distributed computing context. The design and implementation of the **tm.plugin.dc** package is discussed in Section 4.4. The package provides **tm** with supplemental classes and methods in order to benefit from the MapReduce distributed programming model. Additionally, we show how distributed storage can be utilized to facilitate parallel processing of corpora. In Section 4.5 we present the results of a benchmarking experiment of typical tasks in text mining showing the actual impact on performance in terms of execution time. Section 4.6 gives an application in culturomics, employing a corpus of several gigabytes of articles from a prominent newspaper to analyze how word usage has changed over 20 years. Section 4.7 provides computational details. Finally we conclude the paper in Section 4.8, pointing out directions for future research.

4.2 The tm package

The text mining infrastructure package **tm** has now become the de facto standard for running text mining applications in R since it provides a transparent way to prepare textual data for statistical analysis and offers easy extensibility via well

documented interfaces. In this section we summarize the design (data structures), main features, and important interfaces for extending **tm**. Furthermore, we identify challenges to the standard workflow.

4.2.1 Data structures and process flow

In **tm** the main data structure is a *corpus*, an entity similar to a database holding text documents in a generic way. It can be seen as a container to store a collection of text documents where additional metadata is provided on both the corpus (e.g., date of creation, creator, etc.) and document level (e.g., annotations, authors, language, etc.). So-called *sources* are used to abstract document acquisitions, e.g., files from a hard disk, over the Internet, or by other connection mechanisms. A separate *reader* function specifies how to actually parse each item delivered by the source (like XML or HTML documents). The latter eases the usage of heterogeneous text formats. For example assume we have a collection of text documents stored in a directory on a local hard disk. We can simply use a predefined source like `DirSource()` which delivers its content. For some news stories from Reuters (see Section 4.5.1 for a detailed description) in XML format stored in the directory `Data/reuters` we can construct the corpus in R via

```
R> library( "tm" )
R> corpus <- Corpus( DirSource("Data/reuters"),
+                    list(reader = readReut21578XML) )
```

The function `readReut21578XML()` extracts the actual text content and meta information from the XML document.

Alongside the data infrastructure for acquiring text documents the framework provides tools and algorithms to efficiently work with the documents. Several methods have been implemented to abstract the process of document manipulation. For illustration purposes the **tm** package includes a sample data set containing 50 documents of the Reuters corpus on the topic "Acquisitions" (`acq`). We will use it for demonstration (in particular the sixth document) as it provides easy reproducibility. It can be loaded via

```
R> data( "acq" )
R> inspect( acq[[ 6 ]] )

<<PlainTextDocument>>
Metadata:  15
Content:  chars: 381
```

```
A group of affiliated New York
investment firms said they lowered their stake in Cyclops Corp
to 260,500 shares, or 6.4 pct of the total outstanding common
stock, from 370,500 shares, or 9.2 pct.
    In a filing with the Securities and Exchange Commission,
the group, led by Mutual Shares Corp, said it sold 110,000
Cyclops common shares on Feb 17 and 19 for 10.0 mln dlrs.
 Reuter
```

Typical preprocessing tasks (i.e., data preparation and cleaning) like whitespace removal, stemming or stop word deletion can be applied to individual documents contained in the corpus without difficulty.

Stemming denotes the process of deleting word suffixes to retrieve their radicals, i.e., a word stem (also known as root in linguistics). It typically reduces the complexity without any severe loss of information. One of the best known stemming algorithm goes back to Porter (1980) describing an algorithm that removes common morphological and inflectional endings from English words. The **tm** function `stemDocument()` provides an interface to the Porter stemming algorithm.

```
R> stemmed <- stemDocument( acq[[6]] )
R> inspect( stemmed )

<<PlainTextDocument>>
Metadata:  15
Content:  chars: 350

A group of affili New York
invest firm said they lower their stake in Cyclop Corp
to 260,500 shares, or 6.4 pct of the total outstand common
stock, from 370,500 shares, or 9.2 pct.
    In a file with the Secur and Exchang Commission,
the group, led by Mutual Share Corp, said it sold 110,000
Cyclop common share on Feb 17 and 19 for 10.0 mln dlrs.
 Reuter
```

Stop words are words that are so common in a language that their information value is almost zero, i.e., they do not carry significant information (van Rijsbergen 1979). Therefore it is a common procedure to remove such stop words. Similarly to stemming, this functionality is already provided by **tm** via the `removeWords()` function. Removal of whitespace (blanks, tabulators, etc.) and removal of punctuation marks (dot, comma, etc.) can be done via the `stripWhitespace()` and `removePunctuation()` functions.

We denote functions modifying the content of individual text documents in a corpus as *transformations*. Another important concept is *filtering* which basically involves applying predicate functions on collections to extract patterns of interest. See Feinerer *et al.* (2008) for more information about transformations and filtering.

Transformations like whitespace removal, stemming or stop word deletion can easily be applied to all documents contained in a corpus using the `tm_map()` function. The following single call suffices to remove all English stop words from the text corpus `acq`.

```
R> removed <- tm_map( acq, removeWords, stopwords("english") )
R> inspect( removed[[ 6 ]] )

<<PlainTextDocument>>
Metadata:  15
Content:  chars: 327

A group  affiliated New York
investment firms said  lowered  stake  Cyclops Corp
 260,500 shares,  6.4 pct   total outstanding common
stock,  370,500 shares,  9.2 pct.
    In  filing   Securities  Exchange Commission,
 group, led  Mutual Shares Corp, said  sold 110,000
Cyclops common shares  Feb 17  19  10.0 mln dlrs.
 Reuter
```

A very common approach in text mining is to break texts into smaller pieces called *tokens*, and use a suitably normalized subset of these as the *terms* representing the text for subsequent computations (see e.g., Section 2.2 in Manning, Raghavan, and Schütze 2008). Such terms are not necessarily words in the sense of Miller (1995), which are strings made from letters in an alphabet.

The package **tm** supports the construction of a so-called *document-term matrix* (DTM) holding frequencies of distinct terms, i.e., the *term frequency* (TF) for each document. When using **tm** DTMs are stored using a simple sparse representation implemented in package **slam** (Hornik *et al.* 2019). DTM construction typically involves preprocessing and counting TFs for each document. An appropriate method to the generic function `DocumentTermMatrix()` is used to *export* such a matrix from a given corpus (the first argument) applying certain preprocessing steps specified via a list of `control` options (the second argument) that should be executed before counting TFs in each document.

```
R> dtm <- DocumentTermMatrix( acq, list(removePunctuation = TRUE,
+                                        stemming = TRUE) )
R> dtm

<<DocumentTermMatrix (documents: 50, terms: 1395)>>
Non-/sparse entries: 3544/66206
Sparsity           : 95%
Maximal term length: 16
Weighting          : term frequency (tf)
```

Note that the order of transformations to be applied on the corpus can play an important role in terms of run time and actual results. Here we first remove punctuation marks and stem the document before counting individual terms.

The obtained DTMs can be manipulated using the functionality provided by package **slam**, in particular using functions such as `col_sums()`, `row_means()`, or `rollup()` for efficient aggregation. For example we can find the terms which occur most often in the corpus as follows.

```
R> library( "slam" )
R> head( sort(col_sums(dtm), decreasing = TRUE) )

  the  said   and  dlrs   for share
  414   186   173   100    91    86
```

4.2.2 Interfaces

Conceptually we want a corpus to support a set of intuitive operations, like accessing each document in a direct way, displaying and pretty printing the corpus and each individual document, obtaining information about basic properties (such as the number of documents in the corpus), or applying some operation on a range of documents. These requirements are formalized via a set of interfaces which must be implemented by a concrete corpus class:

Subset The `[[` operator must be implemented so that individual documents can be extracted from a corpus.

Display The `print` and `summary` methods must convert documents to a format so that R can display them. Additional meta information can be shown via `summary`.

Length The `length` method must return the numbers of documents in the corpus.

Iteration The `tm_map()` function which can be conceptually seen as an `lapply()` has to implement functionality to iterate over a range of documents and applying a given function.

Export For subsequent text analysis it is crucial to define export mechanisms like document-term matrix construction. This allows for using tools other than those provided with **tm**.

The implementation in R via the **tm** package was mainly driven by these conceptual requirements and interface definitions, and is characterized by a virtual corpus class which defines the above set of pre-defined interfaces. Derived corpus classes must implement these in order to support the full range of desired properties. The main advantage of using a virtual class with well-defined interfaces is that instantiated subclasses work with any function aware of the abstract interface definitions but the underlying implementation and representation of internal data structures is completely abstracted.

4.2.3 Challenges

We identified two challenges when using the **tm** framework. Firstly, big data sets, i.e., data sets which do not fit into main memory like corpora with several millions of documents, cannot easily be constructed and thus processed with the basic facilities provided by **tm**. Secondly, iterations over several millions of documents are rather time consuming. For example performing typical preprocessing steps like stemming or stop word removal on raw text documents can become quite expensive in terms of computing time when the corpus is very large.

Fortunately, operations such as applying transformations and filters are highly amenable to parallelization by construction, as they can separately be applied to each document without side effects. Furthermore, as described in Section 4.2.1 *sources* are used to abstract document acquisitions. Although we use different sophisticated mechanisms for corpus construction like using database back ends it is conceptually appealing and possible to allocate the storage in a distributed manner since communication is usually not limited by a single bottleneck. Ideally, even subsets of the original data set (the corpus) are stored physically distributed on several machines (e.g., in a cluster of workstations). This will not only allow us to increase storage space for data (scaling with the number of participating machines) but also reduce communication costs for parallel computation since only those documents stored locally on a given machine are to be processed on the respective system. Thus, we can use these two approaches (parallel processing, distributing data) to tackle the challenges indicated above.

Both requirements are often fulfilled by well-established distributed programming models such as MapReduce (Dean and Ghemawat 2008). Typically, MapReduce is used in combination with another important building block: the distributed file system (DFS, Ghemawat *et al.* 2003). This approach readily enables and takes care of data distribution and suitable parallel processing of parts of the data in a functional programming style (Lämmel 2007). Given that the MapReduce model fits to the workflow presented above and corresponding open source software libraries are available, it seems an excellent choice when we need to process large corpora in text mining scenarios.

4.3 Distributed computing using MapReduce

In this section we describe the programming model and the corresponding implementation (i.e., the underlying software framework) we employed to achieve parallel text mining in a distributed context. MapReduce is a software framework/library originally proposed by Google for large scale processing of data sets. It consists of two important primitives related to concepts from functional programming, namely a `map` function and a `reduce` function. Basically, the `map` function processes a given set of input data (e.g., collections of text files, log files, web sites, etc.) to generate a set of intermediate data which may/is to be aggregated by the `reduce` function.

Note however, that as pointed out in Lämmel (2007) `map` and `reduce` operations in the MapReduce programming model do not necessarily follow the definition from functional programming. It rather aims to support computation (i.e., map and reduction operations) on large data sets on clusters of workstations in a distributed manner. Provided each mapping operation is independent of the others, all maps can be performed in parallel. Indeed, we can express many tasks in text mining in this model, e.g., preprocessing tasks like stemming.

4.3.1 Programming model

Usually, in this model we consider a set of workstations (nodes) connected by a communication network. Given a set of input data we want to employ these nodes for parallel processing of suitable subsets of the input. *Data locality* is exploited by distributing the data in such a way that parts of the data can efficiently be processed locally on the nodes by individual `map` and aggregated by `reduce` operations. Figure 4.1 shows this conceptual flow of `map`/`reduce` operations on a given data set.

The `map` function usually transforms its input data into a list of key/value pairs. The MapReduce library takes care of reading the corresponding subset of the data

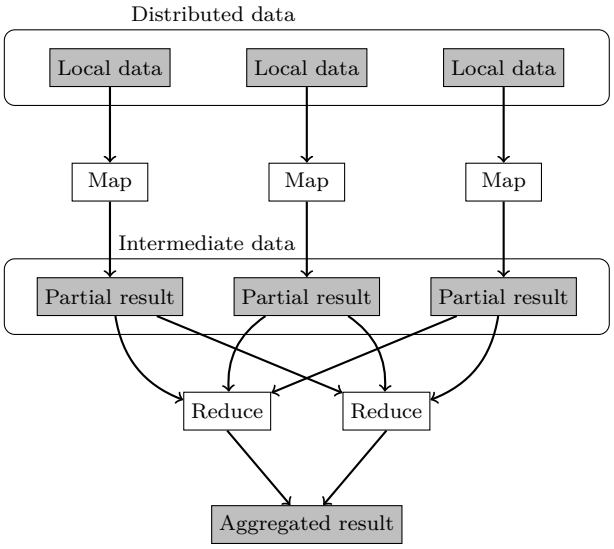

Figure 4.1: Conceptual flow of data processing when using `map` and `reduce` operations. Each node represents a workstation where operations (white boxes) are applied to local parts of data (grey boxes).

located on each node (local data) from disk and handling the results retrieved from the `map` operation (intermediate data). Basically we have three choices to handle intermediate data. First, we can write the processed data to disk. Second, we can apply another `map` function to the intermediate data and/or third, we can apply the `reduce` function which takes the resulting list of key/value pairs and typically aggregates this list based on the keys. This aggregation results in a single key/value pair for each key. All of these operations can be parallelized over the nodes. Typically, every node applies the same `map` function (the mapper) on its local set of data and some of these nodes aggregate different partial result sets (intermediate data) based on the same `reduce` function (the reducer). However, whereas a mapper typically operates on a single data subset, a reducer may aggregate several partial result sets. All of these operations can be chained.

The major advantage of this so-called data parallelism is that if implemented well this approach theoretically scales over any number of nodes. Furthermore, assigning more than one `map` task to each node advances load balancing since the workload generated by all MapReduce operations is not only distributed across multiple computers or a computer cluster but also as data subsets (which typically correspond to the total number of `map` operations) which can be easily relocated to other nodes by the MapReduce library to avoid overload.

To apply the MapReduce programming model in a distributed text mining context we rely on the open source implementation **Hadoop**.

4.3.2 Distributed file system

Implementations of MapReduce like the open source software library **Hadoop** (The Apache Software Foundation 2019) are typically coupled to a distributed file system (DFS) which assists in data distribution and enables fault tolerance. E.g., Google offers the Google File System (GFS) to store data as well as intermediate and final results in a distributed manner. Such a file system assists in exploiting data locality which makes parallel execution of tasks faster as additional communication overhead is avoided. Only pieces of data local to each node are considered for processing which improves overall I/O bandwidth. Furthermore, data is replicated on multiple nodes so that failures like crashed nodes or network routing problems will not affect computation. This enables automatic recovery of data in case of system failures. Such a fault tolerant environment is well suited for large clusters of commodity machines—the prime platform for many scientific institutions and companies because of its cost-effectiveness (Barroso, Dean, and Hölzle 2003).

Similar to GFS the **Hadoop** Distributed File System (HDFS, Borthakur 2010) provides such a scalable and fault tolerant architecture. Copying data to the HDFS implies that the given input data is split into parts (physical blocks or separate files). These parts—often called *chunks*—are distributed over the system and replicated over a predefined number of machines. Files are organized hierarchically in directories and identified by path names.

4.3.3 Software packages

In addition to the MapReduce implementation for distributed manipulation of large data sets and the HDFS, the **Hadoop** framework also includes a utility called ***Hadoop*** *Streaming* implementing a simple interface to **Hadoop** allowing for the usage of MapReduce with any executable or script as the mapper and the reducer. It transforms input data stored on the HDFS and aggregates the results based on the provided scripts.

Several R packages offer functionality based on the MapReduce programming model and/or interface **Hadoop** (Streaming). Package **HadoopStreaming** (Rosenberg 2010) assists in writing proper MapReduce scripts in order to operate on data in a streaming fashion. Similar functionality is offered by the package **mapReduce** (Brown 2011) closely following the framework and nomenclature proposed by Dean and Ghemawat (2008). However, neither package provides facilities for directly interacting with MapReduce libraries. Package **hive** (Theußl and Feinerer 2019) allows the creation of executable R scripts (Rscript, R Core Team 2019a) from provided R functions and automatically run them on a **Hadoop** cluster via **Hadoop** Streaming. This approach offers high level access from within R to the

Hadoop MapReduce and file system functionality. See Appendix D for installing and configuring **Hadoop** and examples on how to use **hive**.

The RHadoop project offers packages **rmr2** (Revolution Analytics 2015) and **rhdfs** (Revolution Analytics 2013) among others. These packages provide a **Hadoop** Streaming connector for running MapReduce tasks in R and functions to access the HDFS. However, they have not been made available in one of the standard R repositories thus far.

4.4 Design and implementation

The bridge between MapReduce libraries and the **tm** infrastructure is characterized by two main design concepts: *distributed storage* and *parallel computation*. The one side of the bridge is designed in such a way that it provides a corpus implementation of the abstract interfaces as outlined in Section 4.2.2. Its classes and methods can transparently be used in combination with the existing **tm** infrastructure. Via the other side of the bridge we can access and modify data stored on the **Hadoop** Distributed File System (HDFS). Data is processed efficiently in parallel using the **Hadoop** Streaming utility.

In this section we discuss the design of the package **tm.plugin.dc** (Theußl and Feinerer 2015) building on functionality provided by interfaces to **tm** and to MapReduce environments. We show that by selecting appropriate building blocks we are not only able to employ the tools provided by **Hadoop** but also any abstract registered (distributed) storage and parallel computing environment. Thus, the implementation realized in **tm.plugin.dc** is driven by *Distributed Storage and Lists* as provided with package **DSL** (Theußl and Feinerer 2020) since it implements both concepts and meets the interface requirements. We use this approach and introduce a new distributed corpus class along with corresponding methods which allow us to analyze large corpora seamlessly without knowing how to use the underlying components of MapReduce or other libraries.

4.4.1 Distributed storage

Typically, a corpus in **tm** is built by constructing an appropriate data structure holding a sequence of single text documents enriched with metadata which further describes textual content and thus offers valuable insights into the document structure. The content for each text document is acquired via source access and copied into main memory. When applied to large corpora computations may slow down significantly due to high RAM usage as there is a practical limit on the maximum corpus size (by the physical memory size minus overhead by the operating system

and other applications).

As pointed out in Section 4.2.3 we can use MapReduce libraries like **Hadoop** to tackle the challenges which come upon us when dealing with large data sets. Thus, in order to use **Hadoop** Streaming from the **Hadoop** runtime environment, we need the corresponding data to be written to the HDFS. We do this in form of key/value pairs generated for each element in the corpus (i.e., for the individual text documents) as required by the MapReduce paradigm. More generally, a corpus using such a (distributed) storage backend can easily be described as a *list* of key/value pairs written to files with the document ID as the key and the corresponding content (a serialized R object) as the value, respectively. We refer to these collections of key/value pairs contained in files as chunks. In R one must only store unique pointers identifying the individual chunks containing the serialized documents.

Both concepts, distributed storage and lists, are implemented in package **DSL**. The S3 class '`DStorage`' defines a virtual storage where files are kept on a file system which possibly spans over several workstations. Typically, data is distributed automatically among these nodes when using such a file system. Objects of class '`DStorage`' "know" how to use the corresponding file system by supplied accessor and modifier methods. The following file system types are supported:

`"LFS":` the local file system. This type uses functions and methods from the packages **base** and **utils** delivered with the R base distribution to handle files.

`"HDFS":` the **Hadoop** Distributed File System. Functions and methods from the **hive** package are used to interact with the HDFS.

This abstract storage class is mainly used for storing key/value pairs as described above. For efficiency reasons several key/value pairs are put line by line into files of a certain maximum size. Indeed, frameworks like **Hadoop** benefit from such a setup (see Section *Data Organization* in Borthakur 2010). Moreover, package **DSL** allows one to store a set of so-called *revisions* for each operation on objects stored in '`DStorage`'. This enables extremely fast switching between various snapshots of the same objects (like a history with rollback feature known from database systems). Using the term "revision" is mainly motivated by the Subversion (SVN, Pilato, Collins-Sussman, and Fitzpatrick 2009) revision concept.

In order to construct a distributed storage object in R the `DStorage()` function from package **DSL** is used. This constructor takes the following arguments:

`type:` the file system type,

`base_dir:` the directory under which chunks containing key/value pairs are to be stored,

`chunksize:` the maximal size of a single chunk,

`keep:` specifying whether to keep data of all stages in a processing chain as revisions.

In the following example we instantiate a distributed storage of `type "HDFS"` using the system-wide or a user-defined *temporary directory* as the base directory (`base_dir`) and a chunk size of 10MB.

```
R> library("DSL")
R> ds <- DStorage( type = "HDFS", base_dir = tempdir(),
+                  chunksize = 10 * 1024^2 )
R> ds

DStorage.
- Type: HDFS
- Base directory on storage: /tmp/RtmpzU8pKl
- Current chunk size [bytes]: 10485760
```

Distributed lists are defined by the S3 class '`DList`'. Objects of this class behave similar to standard R lists but use a distributed storage of class '`DStorage`' to store their elements. Distributed lists can easily be constructed using the function `DList()` or be obtained via coercion using the generic function `as.DList()`. Available methods support coercion of R lists and character vectors representing paths to data repositories as well as coercion of '`DList`' objects to lists.

```
R> dl <- DList( letters = letters, numbers = 0:9  )
R> dl

A DList with 2 elements
```

```
R> l <- as.list( letters )
R> names(l) <- LETTERS
R> dl <- as.DList(l)
R> identical( as.list(dl), l )

[1] TRUE
```

The above example uses the default storage type, namely `"LFS"` for storing list elements. In order to set a user defined storage the `DStorage` argument to the `DList()` constructor is used.

```
R> dl <- DList( letters = letters, numbers = 0:9, DStorage = ds )
```

Furthermore, we can replace the storage assigned to a distributed list. The data is automatically copied to the new storage.

```
R> dl <- DList( letters = letters, numbers = 0:9 )
R> DL_storage(dl)

DStorage.
- Type: LFS
- Base directory on storage: /tmp/RtmpzU8pKl
- Current chunk size [bytes]: 10485760

R> DL_storage(dl) <- ds
R> DL_storage(dl)

DStorage.
- Type: HDFS
- Base directory on storage: /tmp/RtmpzU8pKl
- Current chunk size [bytes]: 10485760
```

4.4.2 Parallel computation

Once we have documents stored on the distributed storage as distributed lists, we want to perform computations on the data pieces local to each processing node. Such computations are highly parallel and scale with the number of available workstations. A recurrent function when computing on lists in R is `lapply()` and variants thereof. Conceptually, this is similar to a `map` function from functional programming where a given (R) function is applied to each element of a vector (or in this case a list). Other typical operations on distributed data are *collective* operations. Functions of this type commonly gather or aggregate data based on a given set of instructions. In functional programming the latter is called `reduce` but possible variations also exists in other areas (e.g., in the MPI standard, see Message Passing Interface Forum 1994, 2003).

Package **DSL** offers the following high-level collective and apply-style functions:

`DGather()`: this collective operation is inspired by MPI_GATHER defined in Message Passing Interface Forum (2003). However, instead of collecting results from processes running in parallel, `DGather()` collects the contents of chunks holding the elements of a '`DList`'. By default a named list of length the number of chunks is to be returned. Its elements are character vectors of values from key/value pairs stored in chunks read line by line from the corresponding chunk. Alternatively, `DGather()` can be used to retrieve the keys only.

`DLapply()`: is an (l)apply-type function which is used to iteratively *apply* a function to a set of input values. In case of `DLapply()` input values are elements of '`DList`' objects (i.e., the value of a key/value pair). A distributed list of the same length is to be returned.

`DMap()`: is the more general variant of `DLapply()` above where both the key and the value from a key/value pair are taken as input. Thus, keys can also be modified. Moreover, `DMap()` may return an object whose length differs to the original as opposed to `DLapply()`.

`DReduce()`: this collective operation takes a set of (intermediate) key/value pairs and combines values with the same associated key using a given directive (the `reduce` function). By default values are concatenated using the `c()` operator.

Parallel execution of the above operations on a '`DList`' object is ensured by the parallel environment which is associated with the assigned distributed storage. In order to take advantage of the MapReduce parallel computing paradigm for operations on '`DList`' the HDFS storage type must be used which has **Hadoop** Streaming associated. Package **DSL** depends on **hive** for rewriting `map` functions on-the-fly to **Hadoop** `map` functions by creating executable R scripts which are sent to the **Hadoop** environment via the **Hadoop** Streaming utility. In detail, every node is assigned a particular chunk located in the HDFS. As indicated in Section 4.3.2 this approach allows for low communication overhead as each node typically accesses and computes only on the data physically located at its position. Each list element located in the corresponding chunk gets unserialized and the `map` function is applied. The (again serialized) results are stored on the HDFS and the pointers in the '`DList`' are updated to match the corresponding chunks. This approach allows us to use not only single (multi-core) systems but also highly scalable clusters of workstations.

For LFS-based storage types the associated parallel environment is *multicore* as provided by the **parallel** package (R Core Team 2019a) available as part of R since version 2.14.0 (see the package vignette for details). This simpler technology can be used without extra configuration on most systems to store and process corpora not fitting into main memory. However, it does not scale beyond the boundaries of a single (possibly multi-core) system.

4.4.3 The distributed corpus class

Since **tm** corpora are basically lists of objects of class '`TextDocument`' enriched with metadata it seems only natural to encapsulate this storage abstraction in distributed lists as provided with package **DSL**. Appropriate methods for '`DList`' objects ensure

that the local files delivered by a source instance are transparently loaded into the distributed storage as a list of key/value pairs.

We call a corpus based on objects of class '`DList`' a *distributed corpus*. This new type of corpus implemented in package **tm.plugin.dc** reduces the main memory consumption drastically since even for millions of documents we keep only the pointers to the (serialized) documents in memory, which occupy just a few megabytes. Technically, we implemented the class '`DCorpus`' which inherits from a standard corpus on the one hand and from class '`DList`' on the other hand. Building the bridge in this way allows us to utilize such a corpus instance in all use cases of a standard corpus by directly employing corresponding high-level collective and apply-style functions like `DGather()`, `DMap()` and `DReduce()`. Since the **tm** infrastructure is designed in a very modular and generic way as described in Section 4.2.2, we only needed to write methods for a few generic functions in order to abstract the underlying distributed storage.

Transformations triggered via `tm_map()` are applied to elements of '`DCorpus`' objects simply by using `DLapply()` from **DSL** instead of `lapply()` defined in the default method in **tm**. The construction of document-term matrices (DTMs) is a combination of `map` and `reduce` steps.

```
R> intermed <- DReduce(DMap(x, map), reduce)
```

In the `map` step preprocessing as described above is applied to the given corpus `x` and the remaining terms are counted and stored so that a term represents the key and the value is a list of document ID and term frequency. The `reduce` step then aggregates (concatenates), for each term, ID and the corresponding term frequency. The result delivered with `DReduce()` is stored as intermediate data (`intermed`). All of these steps may run in parallel. Eventually, based on `intermed` the DTM is constructed from the individual term vectors read from the distributed storage via `DGather()` and combined on the master node.

This implementation allows us to seamlessly use '`DCorpus`' objects in all scenarios supported by the **tm** package and beyond that to hold large amount of textual data as distributed corpora in R.

4.4.4 Using package tm.plugin.dc

The package **tm.plugin.dc** has to be attached in order to make use of the class '`DCorpus`'.

```
R> library( "tm.plugin.dc" )
```

In order to take advantage of the MapReduce parallel computing paradigm for operations on '`DCorpus`' we need to specify to use the HDFS storage type which has

Hadoop Streaming associated. This requires a working Hadoop installation (e.g., on a cluster of workstations) and package **hive** installed which is loaded automatically in the background. **hive** offers an interface to file system accessors and high-level access to Hadoop Streaming. The function `DStorage()` is used to prepare the corresponding storage to be used for text mining tasks.

```
R> storage <- DStorage( type = "HDFS", base_dir = "/tmp/dc" )
```

Similar to the standard process flow presented in Section 4.2.1 the data has to be retrieved from the specified source via

```
R> dc <- DCorpus( DirSource("Data/reuters"),
+                 list(reader = readReut21578XML), storage )
```

or we can *coerce* class 'Corpus' to 'DCorpus' by using the generic `as.DCorpus()`.

```
R> data( "acq" )
R> dc <- as.DCorpus( acq, storage )
R> dc
```

```
<<DCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 50
```

Both functions take a pre-defined storage object as argument specifying that data (either delivered by the source or contained in the corpus) is to be stored as chunks on the given (distributed) storage (see Section 4.4.1). After that, appropriate methods ensure that 'DCorpus' can be handled as defined for 'Corpus'. This allows for a seamless integration of the HDFS or any other storage type defined in **tm.plugin.dc** into **tm** without changing the user interface. For example calling `tm_map()` has the same effects as shown in Section 4.2.1 but uses the Hadoop framework for applying the provided `map` functions instead.

```
R> dc <- tm_map( dc, stemDocument )
R> stemmed <- tm_map( acq, stemDocument )
R> all( sapply(seq_along(acq), function(x)
+      identical(dc[[ x ]], stemmed[[ x ]])) )
```

```
[1] TRUE
```

The processed documents are still stored on the HDFS since by concept the return value of `tm_map()` must be of the same class as the input value. They can easily be retrieved with corresponding accessor methods.

Furthermore, since 'DStorage' offers to store revisions of contained objects, we are able to switch between various snapshots of the same corpus. This enables methods to work on different levels in a preprocessing chain, e.g., before and after stemming. In addition revisions allow for backtracking to earlier processing states, a concept similar to rollbacks in database management systems. Revisions are enabled by default and can be retrieved or set using the functions `getRevisions()` and `setRevision()`, respectively.

```
R> revs <- getRevisions( dc )
R> revs
```

```
[1] "DSL-20191123-102316-eibpbnfmgf" "DSL-20191123-102316-nvgwrfdtvj"
```

```
R> dc <- setRevision( dc, revs[ length(revs) ] )
R> all( sapply(seq_along(acq),
+              function(x) identical(dc[[ x ]], acq[[ x ]])) )
```

```
[1] TRUE
```

The first element in the revision vector always represents the most recent revision (such as the resulting corpus after applying transformations) and the last element represents the revision of the original corpus. Revisions can be turned off using

```
R> keepRevisions( dc ) <- FALSE
```

Another method for 'DCorpus' ensures that the DTM is constructed in parallel. Based on the storage (here HDFS) the Hadoop Streaming utility is used to construct both, the term vectors per document (the `map` step) and triplets of the form $(term, ID, tf)$ (the `reduce` step). Finally, after all `map` and `reduce` steps succeeded in their respective task the resulting matrix is constructed from the aggregated data stored in the HDFS on the master node. Note that the Hadoop runtime environment allows us to set the number of parallel working reducers. This can be done via the function `hive::hive_set_nreducer()`. Otherwise only one reducer will be used.

```
R> DocumentTermMatrix( dc, list(stemming = TRUE,
+                               removePunctuation = TRUE) )
```

```
<<DocumentTermMatrix (documents: 50, terms: 1521)>>
Non-/sparse entries: 3633/72417
Sparsity           : 95%
Maximal term length: 16
Weighting          : term frequency (tf)
```

4.5 Performance

In this section we illustrate the performance of the distributed corpus implementation in two experiments. In the first experiment we study the runtime behavior of typical text mining tasks when using the plug-in package introduced in Section 4.4.3. We compare the results to the parallel computing approach implemented in the **tm** package. The latter uses the *Message Passing Interface* (MPI, Message Passing Interface Forum 1994, 2003) and corresponding R interface packages for parallel processing.

In the second experiment we investigate the performance of our '`DCorpus`' implementation when using corpora showing different characteristics. In particular we are interested in the runtime and throughput behavior when varying corpus size or document size.

4.5.1 Data

For our performance experiments we consider four corpora: the *Reuters-21578* corpus, a collection of *Research Awards Abstracts* from the National Science Foundation (NSF), the *Reuters Corpus Volume 1* (RCV1) and the *New York Times (NYT) Annotated Corpus*. Table 4.1 gives an overview on the different corpora by showing the following figures: the number of documents included in the corpus (the length of the corpus), the mean number of characters per document (the mean document length), and the disk space needed to store the corpus (the corpus size).

Pre-built data packages for the freely redistributable Reuters-21578 and NSF corpora (**tm.corpus.Reuters21578** and **tm.corpus.NSF**, respectively) can be downloaded from the data repository of the Institute for Statistics and Mathematics of the WU Wirtschaftsuniversität Wien (`https://datacube.wu.ac.at`). We only provide corpus packages for Reuters-21578 and NSF due to license restrictions. Packages for the RCV1 and NYT corpora can be obtained from the author if the right to use the data can be verified.

Reuters-21578 The Reuters-21578 data set (Lewis 1997) contains stories collected by the Reuters news agency. The data set is publicly available and has been widely used in text mining research within the last decade. It contains 21,578 short to medium length documents in XML format (obtainable e.g., from `http://ronaldo.cs.tcd.ie/esslli07/data/`) covering a broad range of topics, like mergers and acquisitions, finance, or politics. To download the corpus use:

```
R> install.packages( "tm.corpus.Reuters21578",
+            repos = "https://datacube.wu.ac.at", type = "source" )
```

	No. of docs	Mean no. of char. / doc[1]	Corpus size [MB][2]
Reuters-21578	21,578	736.39/834.42	87
NSF	51,760	2,895.66	236
RCV1	806,791	1,426.01	3,804
NYT	1,855,658	3,303.68/3,347.97	16,160

Table 4.1: Number of included documents, average number of characters per document, and uncompressed size on the file system for each corpus.

```
R> library( "tm.corpus.Reuters21578" )
R> data( "Reuters21578" )
```

NSF Research Awards Abstracts This data set consists of 129,000 plain text abstracts describing NSF awards for basic research submitted between 1990 and 2003. The data set can be obtained from the *UCI Machine Learning Repository* (http://archive.ics.uci.edu/ml/). The corpus is divided into three parts. We used the largest part (*Part 1*) in our experiments.

Reuters Corpus Volume 1 Lewis *et al.* (2004) introduced the RCV1 consisting of about 800,000 (XML format) documents as a test collection for text categorization research. The documents contained in this corpus were sent over the Reuters Newswire (https://www.reutersagency.com/en/products/newswires/) during a 1-year period between 1996-08-20 and 1997-08-19. RCV1 covers a wide range of international topics, including business & finance, lifestyle, politics, sports, etc. The stories were manually categorized in three category sets: topic, industry and region.

NYT Annotated Corpus The largest data set in our experiment contains over 1.8 million articles published by the New York Times between 1987-01-01 and 2007-06-19 (Sandhaus 2008). Documents and corresponding metadata are provided in an XML like format: News Industry Text Format (NITF).

4.5.2 Procedure

The first experiment consists of running several preprocessing steps and constructing DTMs which usually constitute the major computation effort. With the help of the new class '`DCorpus`' delivered with the **tm.plugin.dc** package we can transparently use **Hadoop**, or more specifically, the HDFS, as "extended" memory to store corpora. Parallelization of transformations (via `tm_map()`) and DTM construction (via `DocumentTermMatrix()`) is then supported by appropriate methods

[1] with/without considering empty documents
[2] calculated with the Unix tool `du`

using the **Hadoop** Streaming utility (see Section 4.4.3). In this experiment we measure the runtime behavior of individual tasks using a selected number of CPUs in order to demonstrate the performance gain in terms of speedup. Moreover, since **tm** supports parallelization of transformations, filter operations, and DTM construction via a `parLapply` backend we can use the results obtained in this approach to benchmark our '`DCorpus`' implementation.

The `parLapply()` approach as implemented in the **tm** infrastructure is mainly motivated by the fact that most operations on the documents are independent from the results of other operations. As a consequence there is plenty of room for parallelization, i.e., parallel execution of code fragments on multiple processors with relatively small overhead. This is especially of interest since hardware performance gains during the last years mainly stem from multi-core or multi-processor systems instead of faster (e.g., higher clock frequency) single core processors. Support can easily be activated via `tm_parLapply_engine()` and providing to this function the initialized '`cluster`' object, e.g., created with functionality from the **parallel** package. An earlier version of **tm** internally used the **snow** (Tierney, Rossini, Li, and Sevcikova 2018) package to manage an MPI cluster which in turn delegated the parallel execution triggered via the `parLapply()` function to the **Rmpi** package (Yu 2002, 2018).

The code stays relatively simple since by design the parallel execution of `lapply()` operations is ensured (e.g., via `parLapply()`). This enables the usage on multi-core systems (via multiple instances on individual cores running on a single physical workstation) and on multiprocessor systems (via accessing multiple physically distributed machines). In any case, `parLapply()` splits up the input corpus into a set of suitable chunks of documents and distributes them on the cluster. Then the chunks get processed by the individual participating nodes and results are collected.

However, on the master node, i.e., the node running the controlling R instance of a cluster, the whole data set stays in the main memory while parts of the data are being processed in parallel on the worker nodes (running R processes for executing operations defined via `parLapply()`). Thus, in contrast to the MapReduce approach the **tm**/`parLapply()` implementation is limited by the main memory of the calling machine in terms of data set size. Even if we are able to run several steps in parallel, the whole corpus has to be loaded into RAM initially. As a consequence we are limited to using the Reuters-21578 and the NSF Research Awards Abstracts (Part 1) corpora.

In the second experiment we export DTMs from all four corpora in order to investigate how runtime and throughput (measured as the number of processed characters or bytes per second) is affected with respect to the different characteristics of the corpora employed. Timings also consider the full preprocessing chain applied

before constructing the DTM.

All individual tasks were repeatedly (three times) run for a selected number of CPUs.

4.5.3 Results

Figure 4.2 shows the runtime improvements achieved in the first experiment where **tm** uses parallelization via MPI (solid line) and **Hadoop** (dashed line). As test set we used the complete Reuters-21578 data set (upper row) and part 1 of the NSF data set (lower row), and performed stemming (left) and stopword removal (middle) for each document in the corpus. We set the number of processor cores available to a range from one to 32. The figure depicts the averaged runtime (three runs per setting) necessary to complete all operations, showing a clear indication how **tm** profits from multi-core or distributed parallelization of typical preprocessing steps on a realistic data set. Both approaches scale almost linearly with the amount of processing cores. Interestingly, MPI scales superlinearly in a few cases, e.g., for stopword removal on the Reuters-21578 data set. One possible explanation is that `gsub()`, which is internally used to replace stopwords with an empty string, takes significantly more time on longer strings (i.e., its asymptotic runtime behavior is bigger than $O(n)$ where n denotes the input string length).

Preprocessing is an embarrasingly parallel computing task, thus we expect a speedup $S(p) = p$ when employing p processing cores. However, Amdahl's law (Amdahl 1967) states that if only a fraction f of a given task can be made parallel, the speedup can be calculated as $S(p) = \frac{1}{(1-f)+f/p}$. Since in Figure 2 speedups for **Hadoop** are not linear but for MPI almost are we infer that the fraction $(1-f)$ must be higher for **Hadoop** than for MPI. Thus, costs for starting the **Hadoop** framework (= overhead) must be higher. This is especially relevant for smaller data sets and for computationally cheap operations. In such cases MPI easily outperforms **Hadoop**. For large corpora and computationally expensive operations the **Hadoop** overhead is negligible which makes **Hadoop** the natural choice for large data sets. On clusters of workstations **Hadoop** additionally addresses typical problems like network transfers (limited bandwidth, low latency) and data handling (automatic data split, global access, redundancy) by using a distributed file system. This makes the MapReduce approach the preferred choice for large corpora compared to the MPI approach where many of these problems must be solved manually.

Results for constructing DTMs are shown in Figure 4.2 on the right for Reuters-21578 and NSF, respectively. Interestingly, MPI outperforms **Hadoop** in this case since communication costs are lower when term vectors are transferred. Nevertheless, the MPI approach seems to be quite unstable since speedups do not always

increase when adding a core. Furthermore, we did not manage to get results when using 32 cores probably due to configuration or network setup issues. But most importantly in contrast to **Hadoop** where data is almost always kept on disk, MPI holds data in memory and thus scalability is limited.

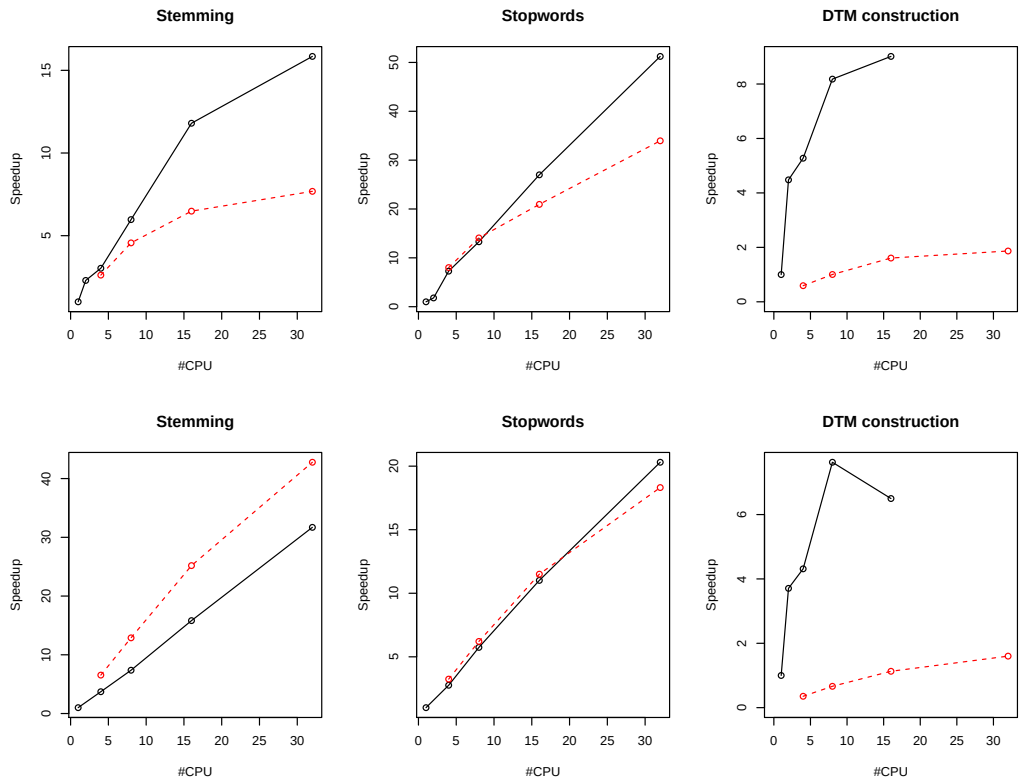

Figure 4.2: Runtime in seconds for stemming, stopword removal, and DTM construction on the full Reuters-21578 data set (upper row) and on part 1 of the NSF data set (lower row) utilizing either **Hadoop** (dashed line) or MPI (solid line) with up to 32 processing cores.

The second experiment only employing the **Hadoop** framework reveals that throughput increases significantly for large data sets as seen in Table 4.2. Here, throughput is measured as the number of 1,000 characters per second ([k char/s]) on the one hand and megabytes per second ([MB/s]) on the other hand. From this table we see some interesting behavior compared to the characteristics of the individual corpora shown in Table 4.1. Corpora containing documents with more content (in terms of the mean document length) are processed faster and the mean document length seems to affect the throughput more than the raw corpus size. One possible explanation for the former behavior is that fewer I/O operations have to be performed compared to the total corpus size since we iterate over the documents in the corpus and not over equal sized text chunks. Furthermore, this might also influence the latter behavior as only for the largest data set we observed a significant gain in throughput where load balancing of **Hadoop** is leveraged and becomes

	runtime [s]	throughput [k char/s]	throughput [MB/s]
Reuters-21578	93	193.6	0.94
NSF	291	515.0	0.81
RCV1	5805	198.2	0.66
NYT	8330	745.8	1.94

Table 4.2: Corpus processing statistics. Constructing DTMs with 32 **Hadoop** nodes on a cluster of workstations.

effective for the given number of nodes.

To sum up, **Hadoop** is the technology of choice for (1) corpora containing documents with lots of text and/or (2) corpora which are large in terms of disk space required.

4.6 Application

It was well established in corpus linguistics (e.g., Francis and Kučera 1982) that (word-type) term frequency distributions obtained from large text collections are typically heavily skewed, with relatively few terms covering most of the texts. Are texts maybe getting increasingly "simple" over time, in the sense of increasing coverage by the basic vocabulary? This is a typical question in the spirit of the new exciting field of culturomics (Michel *et al.* 2011), which deals with the development of human behavior and culture reflected in language and word usage.

In this section we add to this field by investigating how the text coverage in newspaper articles has developed over time. Specifically, we analyze the multitude of text documents published by the New York Times between 1987-01-01 and 2007-06-19. The corresponding corpus consists of 1,855,658 short- to medium-length articles from various genres with a mean of 552 terms per document. As such, this corpus is too large for being handled with the standard text mining tool chain available in R. However, we can use the distributed framework presented in the paper in order to process the corpus quite efficiently. This makes it possible to not only investigate the culturomics question at hand, but also to subject the corpus to a variety of other statistical analyses.

First, we compare the text coverage given a fixed vocabulary size for the NYT corpus with the results of Francis and Kučera (1982) (see also `http://en.wikipedia.org/wiki/Vocabulary`). Let $\mathcal{T}$ be a set of different terms, then text coverage is defined as follows.

$$\text{coverage} = \frac{\text{number of terms in a given text exactly matching a term in } \mathcal{T}}{\text{number of all terms in the text}}$$

In sum we find 1,023,484,418 terms in the whole corpus from which we can derive our vocabulary of 547,400 unique terms. This was conveniently achieved by reading all text documents given in the New York Times corpus into a '`DCorpus`' and subsequently deriving the DTM.

size	coverage	coverage_NYT
1000	0.72	0.75
2000	0.80	0.84
3000	0.84	0.88
4000	0.87	0.91
5000	0.89	0.92
6000	0.90	0.93
15851	0.98	0.97

Table 4.3: Vocabulary coverage in the NYT Corpus compared to standard English text coverage as identified by Francis and Kučera (1982).

Table 4.3 shows that by knowing the 2,000 English words with the highest frequency, one would know on average 80% of the terms in English texts or 84% of the terms in NYT articles. However, text coverage using a given vocabulary is not necessarily stable over time since language use may change (see e.g., Hogg and Denison 2008). To investigate how the active vocabulary has changed over time, we can use the *date of publication* metadata contained in the corpus to easily derive a time series of text coverages for a given vocabulary by suitably aggregating the DTM. We know that language texts could simultaneously get "more complicated" in the sense that far tails of the term frequency distribution get heavier, i.e., that increasingly more terms are needed to cover the remaining words. However, we do not pursue the latter issue here as it requires much more extensive Natural Language Processing (NLP) such as named entity recognition and morphological standardization.

Figure 4.3 shows that text coverage is decreasing almost linearly by roughly 1% over 20 years considering the 1,000 and 4,000 most often used terms in the whole corpus, respectively. Considering this scenario we might conclude that texts are not getting simpler over time.

However, if we consider only stop words, i.e., words that appear in general (in this case English) texts frequently but do not carry significant information, the picture changes. Over the whole NYT corpus we found 117 matching stop words given the list of predefined stop words available in **tm**. The average text coverage using only stop words is 0.41. Figure 4.4 reveals that text coverage driven by **tm**'s stop words dictionary increases by more than 1% in the first decade until 1997 and remains stable over the second (we do not have a satisfactory explanation for this structural break). This suggests that while texts were getting "more specialized", the amount of non-significant content also increased.

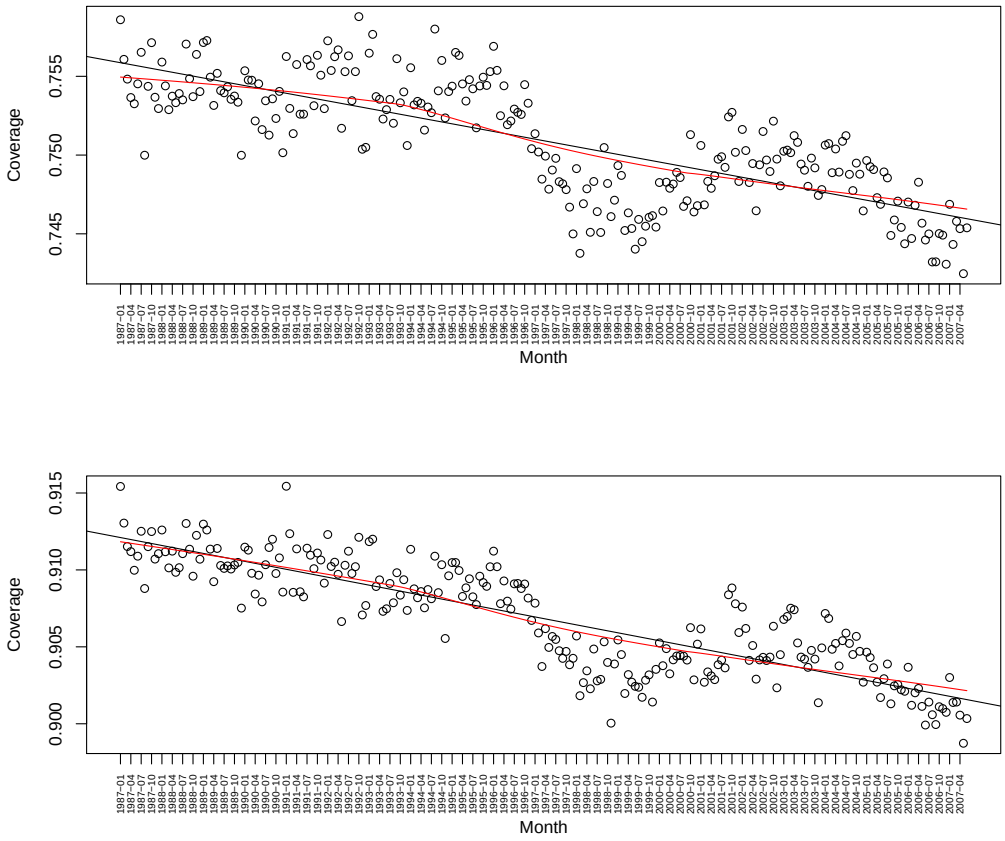

Figure 4.3: Monthly text coverage using top 1000 (top) and 4000 (bottom) terms in the NYT corpus between 1987-01-01 and 2007-06-19.

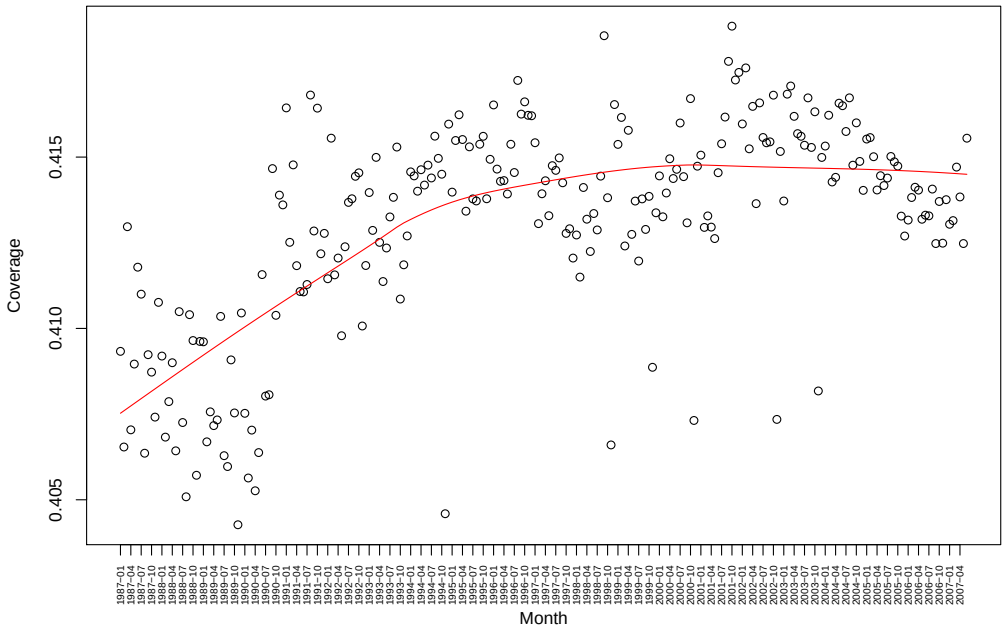

Figure 4.4: Monthly text coverage using stop words in the NYT corpus.

4.7 Computational details

For this dissertation we updated all the packages presented in the original paper (i.e., **DSL**, **hive**, **tm.plugin.dc**) to work with the latest version of R (3.6.2), the latest **Hadoop** implementation (3.2.1), and current state-of-the-art computing environments (Intel Core i7 dual core CPU, 7^{th} generation, 3.5 GHz; 512 GB of fast NVMe SSD storage; 16 GB of main memory). However, all the runtime experiments and applications presented in the paper, in particular the performance experiments presented in Section 4.5 and construction of the DTM for the application in Section 4.6, were not updated and the original results were used. Those experiments were performed on *cluster@WU*, the high performance computing cluster environment of the WU Wirtschaftsuniversität Wien. At the time of the writing of the paper each node of the cluster consisted of an Intel Core 2 Duo CPU 2.4 GHz, 110 GB of local (SATA) hard disk storage reserved for the HDFS, and 4 GB of main memory. All nodes were connected by standard Gigabit Ethernet network in a flat network topology. Parallel jobs had been submitted with the Sun Grid Engine version 6.2 update 3. On cluster@WU the parallel environments **Hadoop** (MapReduce) and **OpenMPI** (MPI) were available in versions 0.20.2 and 1.3.3, respectively. All R code has been executed via version 2.14.0 of R and by using version 0.5-7.1 of the **tm** package.

4.8 Conclusion

The paper has introduced an approach applying the MapReduce distributed programming paradigm to advance feasibility and performance of suitable text mining tasks in R. We showed that distributed memory systems can be effectively employed within this model to preprocess large data sets by adding layers to existing text mining infrastructure packages. We also indicated that data parallelism can very easily be achieved using such an integrated framework without altering the handling of current text mining software available in R (i.e., the **tm** package). This is done via the class '`DCorpus`' implemented in package **tm.plugin.dc**. Appropriate methods make use of a distributed storage (such as the **Hadoop** distributed file system) and a corresponding distributed computing framework (MapReduce). A benchmark experiment showed that applying MapReduce in combination with R on text mining tasks is a very promising approach. The results presented in the paper show a significant performance gain over the sequential code as well as very good scalability when employing the distributed memory model. Such an approach enables us to process large data sets like the NYT corpus and to use R's rich statistical functionalty for large scale text mining applications. We showed this in a culturomics application

scenario.

With the release of package **tm.plugin.dc** we now have two options for handling text corpora in R: using functionality provided with **tm** where data is kept in main memory or using derived corpus classes where data stays on disk and is only accessed when needed. Unfortunately, no general rule exists that defines the most efficient option for a certain text mining application. Of course, depending on corpus size and available RAM on the main workstation, i.e., when corpora get too big, one has to use the distributed corpus implementation presented in the paper. In other scenarios, i.e., for corpus sizes which are manageable with standard tools, it needs to be investigated which approach is the most promising to be employed. Ideally, **tm** would implement suitable heuristics for choosing the best option given a set of criteria (corpus size, network topology, number of computing nodes, overhead induced by the parallel environment, available RAM, etc.) automatically.

Chapter 5

Conclusion and Outlook

In this dissertation we elaborated the design of conceptual and computational frameworks for the R language for statistical computing and graphics for two typical data science use cases: optimization problem solving and large scale text analysis.

Both, the design of a general optimization infrastructure as well as the **tm** plug-in for distributed text mining represent an improvement for R as we address existing problems in data science, constructing and solving optimization problems of different types as well as efficiently processing of very large text corpora, and providing the design of new solutions for them.

Whereas the former design provides us with a consistent framework for constructing and solving optimization problems of different types providing a unified interface to available solvers as well as a modeling mechanism based on R language features, the latter allows to transparently distribute the documents of large (text) corpora on one or several storage entities, apply functions on the subsetted corpus possibly in parallel, and gather results on a cluster of workstations or other (distributed) computing platforms. The implemented artifacts, packages **ROI** and **tm.plugin.dc**, have been evaluated with respect to the validity of the design as well as in real world applications: journal consensus ranking and culturomics.

Still, after more than a decade of research in this field there are further opportunities for future research. **ROI** makes it attractive to add new solvers to the R solver landscape, e.g., to take advantage of recent advances in conic optimization (increase availability), to add additional optimization solvers, read/write functions and additional resources (increase capability), and enables package developers to *plug-in* new solvers quite effortlessly to make use of their highly efficient code (eliminate efficiency detriments). This three enablers (availability, capability, and efficiency) may be good guidelines for future research. Moreover, empirical research questions which make use of methods from the field of optimization can be more conveniently answered with the help of R as we have done e.g., with demonstrating the superiority of the consensus ranking over a simpler approach involving rank averaging when

finding adequate aggregations of a set of single rankings.

Driven mainly by the hype in the field of machine learning, other, rather popular, frameworks have emerged such as **TensorFlow** (Abadi, Barham, Chen, Chen, Davis, Dean, Devin, Ghemawat, Irving, Isard, Kudlur, Levenberg, Monga, Moore, Murray, Steiner, Tucker, Vasudevan, Warden, Wicke, Yu, and Zheng 2016, , `https://www.tensorflow.org/`) and Apache **Spark** (Zaharia, Xin, Wendell, Das, Armbrust, Dave, Meng, Rosen, Venkataraman, Franklin *et al.* 2016, `https://spark.apache.org/`) that meet both design requirements of our second data science use case (large scale text analysis), possibly in a more efficient manner: parallel processing and distributing data. Contemporary text books in data science for R, however, include either sections on text mining with **tm** (such as, e.g., Baumer, Kaplan, and Horton 2017) or sections how to do large scale text analysis with R in general without a dedicated infrastructure for text mining similar to **tm** (such as, e.g., Luraschi, Kuo, and Ruiz 2019), but not including both. Nevertheless, the abstractions *distributed corpus* and *distributed lists* as well as appropriate methods would enable data scientist, statisticians and others conducting their research using R and **tm** to make use of any implemented storage (distributed file system) and any corresponding distributed computing framework, given the bridge to the respective framework is built.

Finally, both artifacts being published on CRAN for several years now, we are also able to make a statement regarding the utility of the developed R packages outside the individual data science use cases, i.e., the development environment, by looking at other community contributions when they directly or indirectly reuse code of the respective package (Theußl *et al.* 2011). A higher number of references to the package would indicate a higher utility for other use cases. Whereas, at the time of this writing, package **ROI** is used by seven other packages on CRAN (this includes all reverse depends, reverse imports, and reverse suggests excluding the **ROI.plugin.*** packages), there are no references to package **tm.plugin.dc**. Clearly, the ambition of providing a general optimization infrastructure for R will have a higher impact than providing a package more focused on a specific use case. Here to note are the packages **ompr** (Schumacher 2018), which builds on top of **ROI** to provide a modeling language inspired by the Jump project in Julia (Lubin and Dunning 2015) to model and solve mixed integer linear programs and **PortfolioAnalytics** (Peterson and Carl 2018), which contains a vignette showcasing various portfolio optimization problems solved with **ROI**.

Appendix A: Tables

	Package	Library	Objective	Constraints	Mixed Integer
1	**alabama**		functional	nonlinear	No
2	**BB**		functional	no	No
3	**cccp**		conic	conic	No
4	**clpAPI**	clp	linear	linear	No
5	**CLSOCP**		conic	conic	No
6	**clue:::sumt**		functional	nonlinear	No
7	**DEoptim**		functional	box	No
8	**dfoptim**		functional	box	No
9	**ECOSolveR**	ECOS	conic	conic	Yes
10	**GenSA**		functional	box	No
11	**glpkAPI, Rglpk**	GLPK	linear	linear	Yes
12	**kernlab:::ipop**		quadratic	linear	No
13	**lbfgsb3**		functional	box	No
14	**LowRankQP**		quadratic	linear	No
15	**lpSolve, lpSolveAPI**	lp_solve	linear	linear	Yes
16	**minqa**		functional	box	No
17	**NlcOptim**		functional	nonlinear	No
18	**nloptr**	NLopt	functional	nonlinear	No
19	**optimx**		functional	box	No
20	**quadprog**		quadratic	linear	No
21	**rcdd**	cddlib	linear	linear	No
22	**Rcgmin**		functional	box	No
23	**Rcsdp**	CSDP	conic	conic	No
24	**Rdsdp**	DSDP	conic	conic	No
25	**rgenoud**		functional	box	No
26	**Rmalschains**		functional	box	No
27	**Rsolnp**		functional	nonlinear	No
28	**Rsymphony**	symphony	linear	linear	Yes
29	**Rvmmin**		functional	box	No
30	**scs**	scs	conic	conic	No
31	**soma**		functional	box	No
32	**stats**		functional	box	No
33	**trustOptim**		functional	no	No
34	**ucminf**		functional	box	No

Table A.1: Overview optimization packages in R.

	Method	Package	Type	Constraint	G	H	J
1	auglag	**alabama**	local	nonlinear	Yes	No	Yes
2	dfsane	**BB**	local	no	No	No	No
3	sumt	**clue**	local	nonlinear	Yes	No	No
4	DEoptim	**DEoptim**	global	box	No	No	No
5	hjkb	**dfoptim**	local	box	No	No	No
6	nmk	**dfoptim**	local	box	No	No	No
7	GenSA	**GenSA**	global	box	No	No	No
8	lbfgsb3	**lbfgsb3**	local	box	Yes	No	No
9	SANN	**stats**	global	no	No	No	No
10	Nelder-Mead	**stats / optimx**	local	no	No	No	No
11	BFGS	**stats / optimx**	local	no	Yes	No	No
12	L-BFGS-B	**stats / optimx**	local	box	Yes	No	No
13	CG	**stats / optimx**	local	no	Yes	No	No
14	nlminb	**stats / optimx**	local	box	Yes	Yes	No
15	nlm	**stats / optimx**	local	no	Yes	Yes	No
16	ucminf	**ucminf / optimx**	local	box	Yes	No	No
17	uobyqa	**minqa / optimx**	local	no	No	No	No
18	newuoa	**minqa / optimx**	local	no	No	No	No
19	bobyqa	**minqa / optimx**	local	box	No	No	No
20	Rcgmin	**Rcgmin / optimx**	local	box	Yes	No	No
21	Rvmmin	**Rvmmin / optimx**	local	box	Yes	No	No
22	spg	**BB / optimx**	local	box	Yes	No	No
23	NlcOptim	**NlcOptim**	local	nonlinear	No	No	No
24	auglag	**nloptr**	local	nonlinear	Yes	No	Yes
25	bobyqa	**nloptr**	local	box	No	No	No
26	cobyla	**nloptr**	local	nonlinear	No	No	No
27	DIRECT	**nloptr**	global	box	No	No	No
28	isres	**nloptr**	global	nonlinear	No	No	No
29	lbfgs	**nloptr**	local	box	Yes	No	No
30	mlsl	**nloptr**	global	box	Yes	No	No
31	mma	**nloptr**	local	nonlinear	Yes	No	Yes
32	nedlermead	**nloptr**	local	box	No	No	No
33	newuoa	**nloptr**	local	no	No	No	No
34	sbplx	**nloptr**	local	box	No	No	No
35	slsqp	**nloptr**	local	nonlinear	Yes	No	Yes
36	stogo	**nloptr**	global	box	Yes	No	No
37	tnewton	**nloptr**	local	box	Yes	No	No
38	varmetric	**nloptr**	local	box	Yes	No	No
39	genoud	**rgenoud**	global	box	Yes	No	No
40	solnp	**Rsolnp**	local	nonlinear	No	No	No

41	malschains	**Rmalschains**	global	box	No	No	No
42	soma	**soma**	global	box	No	No	No
43	trustOptim	**trustOptim**	local	no	Yes	Yes	No
44	DEoptim	**RcppDE**	global	box	No	No	No
45	JDEoptim	**DEoptimR**	global	nonlinear	No	No	No
46	ga	**GA**	global	box	No	No	No
47	mcga	**mcga**	global	box	No	No	No
48	mcga2	**mcga**	global	box	No	No	No
49	psoptim	**pso**	global	box	Yes	No	No
50	psoptim	**psoptim**	global	box	No	No	No
51	cma_es	**cmaes**	global	box	No	No	No
52	cmaes	**cmaesr**	global	no	No	No	No
53	cmaes	**parma**	global	box	No	No	No
54	GAopt	**NMOF**	global	no	No	No	No
55	DEopt	**NMOF**	global	box	No	No	No
56	LSopt	**NMOF**	global	no	No	No	No
57	PSopt	**NMOF**	global	box	No	No	No
58	TAopt	**NMOF**	global	no	No	No	No
59	GrassmannOptim	**GrassmannOptim**	local	no	Yes	No	No
60	lbfgs	**lbfgs**	local	no	Yes	No	No
61	powell	**powell**	local	no	No	No	No
62	ceimOpt	**RCEIM**	local	box	No	No	No
63	subplex	**subplex**	local	no	No	No	No
64	ipoptr	**ipoptr**	local	nonlinear	Yes	Yes	Yes
65	Rdonlp2	**Rdonlp2**	local	nonlinear	No	No	No
66	Rnlminb2	**Rnlminb2**	local	nonlinear	Yes	Yes	No
67	pureCMAES	**adagio**	global	box	No	No	No
68	snomadr	**crs**	local	nonlinear	No	No	No
69	multimin	**gsl**	local	no	Yes	No	No
70	hydroPSO	**hydroPSO**	global	box	No	No	No
71	neldermead	**neldermead**	local	nonlinear	No	No	No
72	cmaOptimDP	**rCMA**	local	nonlinear	No	No	No
73	trust	**trust**	local	no	Yes	Yes	No
74	abc_optim	**ABCoptim**	global	box	No	No	No
75	CEoptim	**CEoptim**	global	no	No	No	No
76	manifold.optim	**ManifoldOptim**	local	no	Yes	Yes	No
77	ALO	**metaheuristicOpt**	global	box	No	No	No
78	DA	**metaheuristicOpt**	global	box	No	No	No
79	FFA	**metaheuristicOpt**	global	box	No	No	No
80	GA	**metaheuristicOpt**	global	box	No	No	No
81	GOA	**metaheuristicOpt**	global	box	No	No	No

82	GWO	**metaheuristicOpt**	global	box	No	No	No
83	HS	**metaheuristicOpt**	global	box	No	No	No
84	MFO	**metaheuristicOpt**	global	box	No	No	No
85	PSO	**metaheuristicOpt**	global	box	No	No	No
86	SCA	**metaheuristicOpt**	global	box	No	No	No
87	WOA	**metaheuristicOpt**	global	box	No	No	No
88	SD	**mize**	local	no	Yes	Yes	No
89	BFGS	**mize**	local	no	Yes	Yes	No
90	SR1	**mize**	local	no	Yes	Yes	No
91	L-BFGS	**mize**	local	no	Yes	Yes	No
92	CG	**mize**	local	no	Yes	Yes	No
93	TN	**mize**	local	no	Yes	Yes	No
94	NAG	**mize**	local	no	Yes	Yes	No
95	DBD	**mize**	local	no	Yes	Yes	No
96	Momentum	**mize**	local	no	Yes	Yes	No
97	n1qn1	**n1qn1**	local	no	Yes	No	No
98	qnbd	**n1qn1**	local	box	Yes	No	No
99	tnbc	**Rtnmin**	local	box	Yes	No	No
100	COBRA	**SACOBRA**	local	box	No	No	No

Table A.2: GPS in R and their capability to handle type, constraint, gradient (G), Hessian (H), Jacobian (J) information.

Appendix B: Code

B.1 Best subset selection

```
R> subset_selection <- function(A, b, k, beta_lb = -1000, beta_ub =
+                               1000, count_intercept = FALSE,
+                               solver = "auto", ...) {
+    control <- list(...)
+    Q <- 2 * t(A) %*% A
+    L <- -2 * t(b) %*% A
+    x <- OP(objective = Q_objective(Q=Q, L=L),
+            bounds = V_bound(li = seq_len(nrow(Q)),
+                             lb = rep.int(-Inf, nrow(Q))))
+    y <- ROI_reformulate(x, "socp")
+    n <- length(objective(x))
+    x <- OP(objective = c(terms(objective(y))$L, double(n)))
+    L <- constraints(y)$L
+    L <- cbind(L, simple_triplet_zero_matrix(nrow(L), n))
+
+    if ( length(beta_lb) == 1L ) beta_lb <- rep.int(beta_lb, n)
+    if ( length(beta_lb) == 1L ) beta_ub <- rep.int(beta_ub, n)
+
+    LB <- cbind(simple_triplet_diag_matrix(-1, n),
+                simple_triplet_zero_matrix(n, 1),
+                simple_triplet_diag_matrix(beta_lb, n))
+    UB <- cbind(simple_triplet_diag_matrix(1, n),
+                simple_triplet_zero_matrix(n, 1),
+                simple_triplet_diag_matrix(-beta_ub, n))
+
+    if (count_intercept) {
+      SUM <- cbind(simple_triplet_zero_matrix(1, n+1),
+                   matrix(1, 1, n))
+    } else {
+      SUM <- cbind(simple_triplet_zero_matrix(1, n+2),
+                   matrix(1, 1, n-1))
+    }
+
```

```
+    constraints(x) <- C_constraint(rbind(L, LB, UB, SUM),
+                                   c(constraints(y)$cones,
+                                    K_lin(n), K_lin(n), K_lin(1L)),
+                                   c(constraints(y)$rhs, double(n),
+                                    double(n), k))
+
+    types(x) <- c(rep.int("C", length(objective(y))),
+                  rep.int("B", n))
+    len <- length(objective(x))
+    bounds(x) <- V_bound(li = seq_len(len),
+                         lb = rep.int(-Inf, len))
+    maximum(x) <- maximum(y)
+
+    if ( isTRUE(control$dry_run) )
+      return(x)
+    z <- ROI_solve(x)
+    head(solution(z), n)
+  }
```

B.2 SON clustering

```
R> convex_clust <- function(A, gamma, solver = "auto",
+                           control = list()) {
+    m <- nrow(A)
+    n <- ncol(A)
+    ncombn <- ( m * (m-1) / 2 )
+    k <- ncombn * (n+1)
+    obj <- c(rep.int(0, n * m), 1/2, rep.int(gamma, ncombn))
+    b <- c(1, -1, 2*as.double(A), rep.int(0, k))
+    L <- simple_triplet_zero_matrix(nrow = 2 + n * m + k,
+                                    ncol = n * m + 1 + ncombn)
+    L[1, n*m+1] <- -1
+    L[2, n*m+1] <- -1
+    L[2+seq_len(n * m), seq_len(n * m)] <- diag(2, n * m)
+
+    ko <- combn(seq_len(m), 2)
+    M <- matrix(seq_len(n*m), m, n, byrow=FALSE)
+    irow <- n * m + 3
+    cones <- K_soc(n*m+2)
```

```
+    for ( i in seq_len(ncol(ko)) ) {
+      L[irow, n * m + 1 + i] <- -1
+      cones <- c(cones, K_soc(n+1))
+      irow <- irow + 1
+      for (j in seq_len(n)) {
+        L[irow, M[ko[,i], j]] <- c(-1, 1)
+        irow <- irow + 1
+      }
+    }
+
+    op <- OP(objective=L_objective(obj),
+             constraints = C_constraint(L=L, cones=cones, rhs=b),
+             bounds = V_bound(ld = -Inf, nobj = nrow(L)))
+    if ( isTRUE(control$dry_run) )
+      return(op)
+
+    x <-  ROI_solve(op, solver, control)
+    matrix(x$solution[seq_len(n*m)], m, n)
+  }
```

B.3 Graphical lasso

```
R> index_to_vech <- function(i, j, n) {
+      ind_to_vech <- function(i, j, n) {
+          if (j > i) return(NA)
+          (j - 1) * n + i - (j - 1) * j / 2
+      }
+      unlist(mapply(ind_to_vech, i, j, MoreArgs = list(n = n),
+                    SIMPLIFY = FALSE, USE.NAMES = FALSE),
+             recursive = FALSE, use.names = FALSE)
+  }

R> ROI_glasso <- function(s, rho, solver = "auto", ...) {
+      stopifnot(nrow(s) > 1)
+      control <- list(...)
+      stm <- simple_triplet_matrix
+      n <- nrow(s); nv <- (n + 1) * n / 2; nij <- choose(n, 2)
+      ndzx <- (2 * n + 1) * 2 * n / 2
+      seqn <- seq_len(n); seqnv <- seq_len(nv)
```

```
+      seqnij <- seq_len(nij)
+
+      obj <- c(as.vector(vech(s + s * lower.tri(s))), double(ndzx))
+      A <- simple_triplet_diag_matrix(-1, nv + ndzx)
+      cones <- K_psd(c(nv, ndzx))
+
+      ij <- combn(seqn, 2)
+      k <- index_to_vech(n + ij[1,], ij[2,], 2 * n)
+      Z_UPPER <- stm(seqnij, nv + k, rep.int(1, nij), nrow = nij,
+                     ncol = nv + ndzx)
+      cones <- c(cones, K_zero(nij))
+
+      k <- index_to_vech(ij[2,], ij[1,], 2 * n)
+      D_DIAG <- stm(seqnij, nv + k, rep.int(1, nij), nrow = nij,
+                    ncol = nv + ndzx)
+      cones <- c(cones, K_zero(nij))
+
+      kd <- index_to_vech(seqn, seqn, 2 * n)
+      kz <- index_to_vech(n + seqn, seqn, 2 * n)
+      EQ_DIAG <- stm(c(seqn, seqn), nv + c(kd, kz),
+                     c(rep.int(-1, n), rep.int(1, n)),
+                     nrow = n, ncol = nv + ndzx)
+      cones <- c(cones, K_zero(n))
+      A <- rbind(A, rbind(Z_UPPER, D_DIAG, EQ_DIAG))
+
+      EQ_X <- stm(c(seqnv, seqnv), c(seqnv, ndzx + seqnv),
+                  c(rep.int(-1, nv), rep.int(1, nv)),
+                  nrow = nv, ncol = ncol(A))
+      cones <- c(cones, K_zero(nrow(EQ_X)))
+      A <- rbind(A, EQ_X)
+      rhs <- double(nrow(A))
+
+      obj <- c(obj, rep.int(-1, n))
+      j <- nv + kd
+      LOG <- stm(c(3 * seqn, 3 * seqn - 2), c(j, ncol(A) + seqn),
+                 rep.int(-1, 2 * n), nrow = 3 * n,
+                 ncol = ncol(A) + n)
+      A <- rbind(cbind(A, simple_triplet_zero_matrix(nrow(A), n)),
+                 LOG)
```

```
+      cones <- c(cones, K_expp(n))
+      rhs <- c(rhs, rep.int(c(0, 1, 0), n))
+
+      rho_matrix <- matrix(rho, n, n)
+      obj <- c(obj, vech(rho_matrix +
+                           rho_matrix * lower.tri(rho_matrix)))
+      NORM <- stm(i = c(seq_len(2 * nv), seq_len(2 * nv)),
+                  j = c(seqnv, seqnv, ncol(A) + seqnv,
+                        ncol(A) + seqnv),
+                  v = c(rep.int(1, nv), rep.int(-1, 3 * nv)),
+                  nrow = 2 * nv, ncol = ncol(A) + nv)
+      A <- rbind(cbind(A, simple_triplet_zero_matrix(nrow(A), nv)),
+                 NORM)
+      cones <- c(cones, K_lin(2 * nv))
+      rhs <- c(rhs, double(2 * nv))
+
+      model <- OP(objective = L_objective(obj),
+                  constraints = C_constraint(A, cones = cones,
+                                             rhs = rhs),
+                  bounds = V_bound(ld = -Inf, nobj = length(obj)))
+
+      if ( isTRUE(control$dry_run) )
+          return(model)
+
+      so <- ROI_solve(model, solver=solver, control)
+      Y <- matrix(0, n, n)
+      Y[lower.tri(Y, diag=TRUE)] <-
+                so$solution[seq_len(n * (n+1) / 2)]
+      Y[upper.tri(Y)] <- t(Y)[upper.tri(Y)]
+      list(w=chol2inv(chol(Y)), wi=Y,
+           errflag=so$status$code, niter=so$message$info$iter)
+  }
```

Appendix C: Abbreviations

Abbreviation	Full Name
LP	Linear Programming
NLP	Nonlinear Programming
MILP	Mixed Integer Linear Programming
MINLP	Mixed Integer Nonlinear Programming
MIQP	Mixed Integer Quadratic Programming
MIQCP	Mixed Integer Quadraticly Constraint Programming
QP	Quadratic Programming
QCP	Quadraticly Constraint Programming
QCLP	Quadraticly Constraint Programming Linear Programming
QCQP	Quadraticly Constraint Programming Quadratic Programming
SDP	Semidefinite Programming
SOCP	Second Order Cone Programming

Appendix D: Installing Hadoop

In this section we describe how to set up the Hadoop framework in (pseudo) distributed operation. This description is based on https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/SingleCluster.html and focuses on the Linux operating system as this platform is suggested on the website for production use. For a more detailed installation instruction and for other platforms than Linux we refer to the above website.

Basic Hadoop environment

Since Java and the command line utility `ssh` are needed for operation both programs have to be installed on all involved machines (on our test platform we used Java version 1.8 update 231 and OpenSSH 7.9). In order to install Hadoop we first downloaded the compressed archive of the framework following the instructions on the release website (https://hadoop.apache.org/releases.html) and then uncompressed the contents to a directory on a file system which is accessible by all machines running Hadoop (subsequently referred to as `HADOOP_HOME`). For example on a single workstation this directory is usually located somewhere on the corresponding local disk. In case of a cluster of workstation, this directory is usually located on a network file system (e.g., NFS). Several components of the Hadoop framework need to know the path to the installation directory, thus it is specified on each machine via the environment variable `HADOOP_HOME` (this variable has to be added to the user's environment on each machine if it is not done automatically).

Subsequently, few to several changes have to be made in configuration files located in the `$HADOOP_HOME/etc/hadoop` directory depending on the desired operating mode. First, the path to a working Java environment has to be specified in `hadoop-env.sh`. Second, since we want to operate the framework in *pseudo distributed* (i.e., use the HDFS facilities on a single workstation) or *fully distributed* mode, one needs to set up the configuration files `core-site.xml`, `hdfs-site.xml`, `mapred-site.xml`, `yarn-site.xml`, and `workers`. An example configuration of a pseudo distributed system can be found at the end of Appendix D. Alternatively, for various Linux distributions pre-configured packages can be obtained e.g., from https://www.cloudera.com/.

To verify the installation one can execute `bin/hadoop version` on the command line (we expect to be in the `$HADOOP_HOME` directory when issueing Hadoop-related commands) which returns the version number of the installed software package.

Hadoop distributed file system

The final step before running Hadoop jobs in a pseudo distributed environment involves enabling passwordless `ssh` to localhost and formatting the HDFS (the default system path in recent Hadoop versions is `/tmp/hadoop-$user.name`). The command `bin/hdfs namenode -format` serves for this purpose.

The configured Hadoop cluster can be started via two alternative approaches. In the first approach one can use the command `bin/start-dfs.sh` on the command line or via daemon startup scripts in `/etc/init.d` (e.g., cloudera packages). This is especially useful if an integration to cluster grid engines is desired in order to automate the startup process. In the second approach the Hadoop cluster can be started directly within R using the `hive_create()` and `hive_start()` functions in **hive**. The resulting '`hive`' object, representing the information about the configured cluster is stored for further use with the help of the function `hive()`. Usually, this is done automatically when the package loads, given that the Hadoop framework is referenced to via the `HADOOP_HOME` environment variable or if the executables are in the `PATH` and configurations are put in `/etc/hadoop` (as is with the cloudera packages). However, there is a known issue with IPv6 (see https://cwiki.apache.org/confluence/display/HADOOP2/HadoopIPv6). Thus, if the Hadoop framework does not start, correctly setting the configuration option `net.ipv6.bindv6only` to `0` in `/etc/sysctl.d/bindv6only.conf` will help.

```
R> require( "hive" )
R> hadoop_home <- Sys.getenv( "HADOOP_HOME" )
R> hive( hive_create(hadoop_home) )
R> hive()

HIVE: Hadoop Cluster
- Avail. datanodes: 1
'- Max. number Map tasks per datanode: 2
'- Configured Reducer tasks: 1

R> summary( hive() )

HIVE: Hadoop Cluster
- Avail. datanodes: 1
'- Max. number Map tasks per datanode: 2
'- Configured Reducer tasks: 1
---
- Hadoop version: 3.2.1
```

```
- Hadoop home/conf directory: /usr/local/share/hadoop-3.2.1
- Namenode: NA
- Datanodes:
'- localhost

R> hive_is_available()

[1] FALSE

R> hive_start()
R> hive_is_available()

[1] TRUE
```

When the Hadoop cluster is up and running one can retrieve Hadoop status and job information by visiting specific websites provided by the built-in web front end, two of them are of higher interest. Assuming that Hadoop is configured to run on `localhost` the websites can be accessed via a standard web browser opening `http://localhost:9870` (Hadoop Overview) and `http://localhost:8088` (YARN), respectively. On the former website one can inspect the configuration of the HDFS and browse through the file system. From the latter one can retrieve information about running, completed or failed jobs. We found the log files showing the error output for the distributed batch jobs very useful as they helped us debugging the R scripts generated by the **hive** package for each mapper and reducer.

The HDFS can be accessed directly in R with `DFS_*()` functions. For a complete reference to implemented `DFS_*()` functions see the **hive** help pages.

```
R> DFS_list( "/" )

[1] "tmp"  "user"

R> DFS_dir_create( "/tmp/test" )
R> DFS_write_lines( c("Hello HDFS", "Bye Bye HDFS"),
+                   file = "/tmp/test/hdfs.txt" )
R> DFS_list( "/tmp/test" )

[1] "hdfs.txt"

R> DFS_read_lines( file = "/tmp/test/hdfs.txt" )

[1] "Hello HDFS"   "Bye Bye HDFS"
```

Hadoop configuration

We recommend to set the following parameters in the corresponding configuration files in order to set up pseudo distributed mode on a single machine with two processors/cores. In case of a standard Hadoop installation as described above the file `workers` has to be created in the `etc/hadoop` directory (containing `localhost`). The files reflect a typical configuration for Hadoop version 3.2.1.

`core-site.xml` contains:

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/home/${user.name}/tmp/hadoop-${user.name}</value>
  </property>
</configuration>
```

`hdfs-site.xml` contains:

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```

`mapred-site.xml` contains:

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>mapreduce.application.classpath</name>
    <value>$HADOOP_MAPRED_HOME/share/hadoop/mapreduce/*:
    $HADOOP_MAPRED_HOME/share/hadoop/mapreduce/lib/*</value>
  </property>
```

```
    <property>
      <name>yarn.app.mapreduce.am.env</name>
      <value>HADOOP_MAPRED_HOME=/usr/local/share/hadoop-3.2.1
       </value>
    </property>
    <property>
      <name>mapreduce.map.env</name>
      <value>HADOOP_MAPRED_HOME=/usr/local/share/hadoop-3.2.1
      </value>
    </property>
    <property>
      <name>mapreduce.reduce.env</name>
      <value>HADOOP_MAPRED_HOME=/usr/local/share/hadoop-3.2.1
       </value>
    </property>
  </configuration>
```

yarn-site.xml contains:

```
  <configuration>
    <property>
      <name>yarn.nodemanager.aux-services</name>
      <value>mapreduce_shuffle</value>
    </property>
    <property>
      <name>yarn.nodemanager.env-whitelist</name>
      <value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,
      HADOOP_CONF_DIR,CLASSPATH_PREPEND_DISTCACHE,
      HADOOP_YARN_HOME,HADOOP_HOME,PATH,LANG,TZ</value>
    </property>
  </configuration>
```

slaves contains the IP address of all data nodes (or `localhost`).

References

Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, Kudlur M, Levenberg J, Monga R, Moore S, Murray DG, Steiner B, Tucker P, Vasudevan V, Warden P, Wicke M, Yu Y, Zheng X (2016). "TensorFlow: A System for Large-Scale Machine Learning." In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283. USENIX Association, Savannah, GA. ISBN 978-1-931971-33-1. URL https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

Alizadeh F, Goldfarb D (2003). "Second-Order Cone Programming." *Mathematical Programming*, **95**(1), 3–51. ISSN 1436-4646. doi:10.1007/s10107-002-0339-5.

Amdahl GM (1967). "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities." In *Proceedings of the April 18–20, 1967, spring joint computer conference*, AFIPS '67 (Spring), pp. 483–485. ACM, New York, NY, USA. doi:10.1145/1465482.1465560.

Andersen MS, Dahl J, Liu Z, Vandenberghe L (2012). *Interior-Point Methods for Large-Scale Cone Programming*, chapter 3, pp. 55–83. MIT Press. ISBN 9780262016469. URL https://www.seas.ucla.edu/~vandenbe/publications/mlbook.pdf.

Andersen MS, Dahl J, Vandenberghe L (2019). *CVXOPT: A Python Package for Convex Optimization, Version 1.2.3*. URL http://cvxopt.org/.

ApS M (2019). *Introducing the **MOSEK** Optimization Suite 9.1.6*. URL https://docs.mosek.com/9.1/intro/index.html.

Azar OH, Brock DM (2008). "A Citation-Based Ranking of Strategic Management Journals." *Journal of Economics & Management Strategy*, **17**(3), 781–802.

Bakir A, Vitell S, Rose G (2000). "Publications in Major Marketing Journals: an Analysis of Scholars and Marketing Departments." *Journal of Marketing Education*, **22**(2), 97–107.

Bancroft DR, Gopinath C, Kovács AM, Rejtö LK (1999). "A New Methodology for Aggregating Tables: Summarizing Journal Quality Data." *Journal of Business Venturing*, **14**(3), 311–319.

Bao X, Sahinidis NV, Tawarmalani M (2011). "Semidefinite Relaxations for Quadratically Constrained Quadratic Programming: A Review and Comparisons." *Mathematical Programming*, **129**(1), 129–157. ISSN 1436-4646. doi:10.1007/s10107-011-0462-2.

Barroso LA, Dean J, Hölzle U (2003). "Web Search for a Planet: The Google Cluster Architecture." *IEEE micro*, **23**(2), 22–28.

Baskerville R, Baiyere A, Gregor S, Hevner A, Rossi M (2018). "Design Science Research Contributions: Finding a Balance between Artifact and Theory." *Journal of the Association for Information Systems*, **19**(5), 358–376. doi:10.17705/1jais.00495.

Bates D, Mullen KM, Nash JC, Varadhan R (2014). ***minqa**: Derivative-Free Optimization Algorithms by Quadratic Approximation*. R package version 1.2.4, URL https://CRAN.R-project.org/package=minqa.

Baumer BS, Kaplan DT, Horton NJ (2017). *Modern Data Science with R*. Chapman and Hall/CRC.

Baumgartner H, Pieters R (2003). "The Structural Influence of Marketing Journals: A Citation Analysis of the Discipline and Its Subareas Over Time." *Journal of Marketing*, **67**(2), 123–139.

Belotti P, Lee J, Liberti L, Margot F, Wächter A (2009). "Branching and Bounds Tightening Techniques for Non-Convex MINLP." *Optimization Methods and Software*, **24**(4-5), 597–634. doi:10.1080/10556780903087124. http://dx.doi.org/10.1080/10556780903087124.

Ben-Tal A, Nemirovski A (2001). *Lectures on Modern Convex Optimization*. Society for Industrial and Applied Mathematics. doi:10.1137/1.9780898718829.

Ben-Tal A, Nemirovski A (2019). "Lectures on Modern Convex Optimization." URL http://www2.isye.gatech.edu/~nemirovs/LMCO_LN.pdf.

Benoist T, Estellon B, Gardi F, Megel R, Nouioua K (2011). "**LocalSolver** 1.x: A Black-Box Local-Search Solver for 0-1 Programming." *4OR*, **9**(3), 299. ISSN 1614-2411. doi:10.1007/s10288-011-0165-9.

Benson SJ, Ye Y (2008). "Algorithm 875: **DSDP5**-Software for Semidefinite Programming." *ACM Transactions on Mathematical Software*, **34**(3), 16:1–16:20. ISSN 0098-3500. doi:10.1145/1356052.1356057.

Berkelaar M (2019). ***lpSolve**: Interface to **lp_solve** v. 5.5 to Solve Linear/Integer Programs*. R package version 5.6.13.3, URL https://CRAN.R-project.org/package=lpSolve.

Berkelaar M, Eikland K, Notebaert P (2016). ***lp_solve** 5.5, Open Source (Mixed-Integer) Linear Programming System*. Version 5.5.2.5, URL http://lpsolve.sourceforge.net/5.5/.

Bertsimas D, King A, Mazumder R (2016). "Best Subset Selection via a Modern Optimization Lens." *The Annals of Statistics*, **44**(2), 813–852. ISSN 0090-5364, 2168-8966. doi:10.1214/15-aos1388.

Bezanson J, Edelman A, Karpinski S, Shah VB (2017). "Julia: A Fresh Approach to Numerical Computing." *SIAM Review*, **59**(1), 65–98. doi:10.1137/141000671.

Bihorel S, Baudin M (2018). ***neldermead**: R Port of the Scilab **neldermead** Module*. R package version 1.0-11, URL https://CRAN.R-project.org/package=neldermead.

Bisschop J, Meeraus A (1982). "On the Development of a General Algebraic Modeling System in a Strategic Planning Environment." In *Applications*, volume 20 of *Mathematical Programming Studies*, pp. 1–29. Springer-Verlag. URL http://link.springer.com/chapter/10.1007/BFb0121223.

Bixby RE (2012). "A Brief History of Linear and Mixed-Integer Programming Computation." *Documenta Mathematica*, pp. 107–121. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.592.1021&rep=rep1&type=pdf.

Blei DM, Lafferty JD (2007). "A Correlated Topic Model of Science." *The Annals of Applied Statistics*, **1**(1), 17–35.

Bonami P, Lee J (2013). ***Bonmin** Users Manual*. Version 1.8, URL https://projects.coin-or.org/Bonmin/browser/stable/1.8/Bonmin/doc/BONMIN_UsersManual.pdf?format=raw.

Borchers B (1999). "**CSDP**, A C Library for Semidefinite Programming." *Optimization Methods and Software*, **11**(1-4), 613–623. doi:10.1080/10556789908805765. http://dx.doi.org/10.1080/10556789908805765.

Borda JC (1781). "Mémoire sur les élections au scrutin." Histoire de l'Académie Royale des Sciences.

Boros E, Hammer PL (2002). "Pseudo-Boolean Optimization." *Discrete Applied Mathematics*, **123**(1–3), 155–225. doi:10.1016/s0166-218x(01)00341-9.

Borthakur D (2010). "HDFS Architecture." *Document on Hadoop Wiki*. URL https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

Boyd S, Vandenberghe L (2004). *Convex Optimization*. Cambridge University Press, New York, NY, USA. ISBN 0521833787.

Bravo HC (2016). ***Rcsdp**: R Interface to the **CSDP** Semidefinite Programming Library*. R package version 0.1.55, URL https://CRAN.R-project.org/package=Rcsdp.

Brooks JP, Dulá JH (2013). "The L1-Norm Best-Fit Hyperplane Problem." *Applied Mathematics Letters*, **26**(1), 51–55.

Brown C (2011). ***mapReduce**: mapReduce — flexible mapReduce algorithm for parallel computation*. R package version 1.2.3, URL http://CRAN.R-project.org/package=mapReduce.

Canty A, Ripley BD (2019). ***boot**: Bootstrap R (S-PLUS) Functions*. R package version 1.3-23, URL https://CRAN.R-project.org/package=boot.

Chambers JM (1993). "Greater or Lesser Statistics: a Choice for Future Research." *Statistics and Computing*, **3**(4), 182–184.

Chares PR (2009). *Cones and Interior-Point Algorithms for Structured Convex Optimization Involving Powers and Exponentials*. Ph.D. thesis, Université Catholique de Louvain. URL http://dial.uclouvain.be/pr/boreal/en/object/boreal%3A28538/datastream/PDF_01/view.

Chen X, Yin X (2019). ***NlcOptim**: Solve Nonlinear Optimization with Nonlinear Constraints*. R package version 0.6, URL https://CRAN.R-project.org/package=NlcOptim.

Chi EC, Lange K (2015). "Splitting Methods for Convex Clustering." *Journal of Computational and Graphical Statistics*, **24**(4), 994–1013. ISSN 1061-8600. doi:10.1080/10618600.2014.948181.

Cleveland WS (2001). "Data Science: an Action Plan for Expanding the Technical Areas of the Field of Statistics." *International Statistical Review*, **69**(1), 21–26.

Conceicao ELT (2016). ***DEoptimR**: Differential Evolution Optimization in Pure R*. R package version 1.0-8, URL https://CRAN.R-project.org/package=DEoptimR.

Condorcet MJA (1785). "Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix." Paris.

Cook WD (2006). "Distance-based and Ad Hoc Consensus Models in Ordinal Preference Ranking." *European Journal of Operational Research*, **172**(2), 369–385. ISSN 0377-2217.

Czyzyk J, Mesnier MP, Moré JJ (1998). "The NEOS Server." *IEEE Journal on Computational Science and Engineering*, **5**(3), 68 – 75.

de Leeuw J, Mair P (2009). "Multidimensional Scaling Using Majorization: SMACOF in R." *Journal of Statistical Software*, **31**(3), 1–30.

Dean J, Ghemawat S (2008). "MapReduce: Simplified Data Processing on Large Clusters." *Communications of the ACM*, **51**(1), 107–113. doi:10.1145/1327452.1327492.

Diamond S, Boyd S (2015). "Convex Optimization with Abstract Linear Operators." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 675–683. IEEE Computer Society. doi:10.1109/iccv.2015.84.

Diamond S, Boyd S (2016). "**CVXPY**: A Python-Embedded Modeling Language for Convex Optimization." *Journal of Machine Learning Research*, **17**(83), 1–5. URL http://jmlr.org/papers/volume17/15-408/15-408.pdf.

Dolan ED (2001). "The NEOS Server 4.0 Administrative Guide." *Technical Memorandum ANL/MCS-TM-250*, Mathematics and Computer Science Division, Argonne National Laboratory.

Domahidi A, Chu E, Boyd S (2013). "**ECOS**: An SOCP Solver for Embedded Systems." In *European Control Conference (ECC)*, pp. 3071–3076. URL https://web.stanford.edu/~boyd/papers/pdf/ecos_ecc.pdf.

Donoho D (2017). "50 Years of Data Science." *Journal of Computational and Graphical Statistics*, **26**(4), 745–766. doi:10.1080/10618600.2017.1384734.

Dubois FL, Reeb D (2000). "Ranking the International Business Journals." *Journal of International Business Studies*, **31**(4), 689–704.

Dykstra RL (1983). "An Algorithm for Restricted Least Squares Regression." *Journal of the American Statistical Association*, **78**(384), 837–842. doi:10.1080/01621459.1983.10477029.

Feinerer I (2018). ***tm**: Text Mining Package*. R package version 0.7-6, URL http://tm.R-Forge.R-project.org/.

Feinerer I, Hornik K, Meyer D (2008). "Text Mining Infrastructure in R." *Journal of Statistical Software*, **25**(5), 1–54. ISSN 1548-7660. URL http://www.jstatsoft.org/v25/i05.

Ferreau HJ, Kirches C, Potschka A, Bock HG, Diehl M (2014). "**qpOASES**: A Parametric Active-Set Algorithm for Quadratic Programming." *Mathematical Programming Computation*, **6**(4), 327–363. doi:10.1007/s12532-014-0071-1.

Ferreau HJ, Potschka A, Kirches C (2017). "qpOASES Webpage." https://www.qpOASES.org/.

Fine S, Scheinberg K (2001). "Efficient SVM Training Using Low-Rank Kernel Representations." *Journal of Machine Learning Research*, **2**(Dec), 243–264. URL http://www.jmlr.org/papers/volume2/fine01a/fine01a.pdf.

Fischetti M, Salvagnin D (2010). "Pruning Moves." *INFORMS Journal on Computing*, **22**(1), 108–119. doi:10.1287/ijoc.1090.0329.

Fishburn PC (1972). *Mathematics of Decision Theory*. Mouton, The Hague.

Forrest J, de la Nuez D, Lougee-Heimer R (2004). ***Clp*** *User Guide*. URL http://www.coin-or.org/Clp/userguide/index.html.

Fourer R, Gay DM, Kernighan B (1989). "AMPL: A Mathematical Programming Language." In SW Wallace (ed.), *Algorithms and Model Formulations in Mathematical Programming*, pp. 150–151. Springer-Verlag, New York, NY, USA. doi:10.1007/978-3-642-83724-1.

Francis WN, Kučera H (1982). *Frequency Analysis of English Usage: Lexicon and Grammar*. Houghton Mifflin.

Franke N, Schreier M (2008). "A Meta-Ranking of Technology and Innovation Management/Entrepreneurship Journals." *Die Betriebswirtschaft (DBW)*, **2008**(2), 185–216.

Freese R (2004). "Automated Lattice Drawing." In *Concept Lattices*, volume 2961 of *Lecture Notes in Computer Science*, pp. 589–590. Springer.

Freund RM (2009). "Introduction to Semidefinite Programming (SDP)." URL http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-251j-introduction-to-mathematical-programming-fall-2009/readings/MIT6_251JF09_SDP.pdf.

Friberg HA (2019). ***Rmosek****: The R-to-**MOSEK** Optimization Interface*. R package version 1.3.5, URL https://CRAN.R-project.org/package=Rmosek.

Friedman J, Hastie T, Tibshirani R (2008). "Sparse Inverse Covariance Estimation with the Graphical Lasso." *Biostatistics*, **9**(3), 432–441.

Fry E, Walters C, Scheurmann L (1985). "Perceived Quality of 50 Selected Journals: Academicians and Practitioners." *Journal of the Academy of Marketing Science*, **13**(2), 352–361.

Fu A, Narasimhan B (2019). ***ECOSolveR**: Embedded Conic Solver in R*. R package version 0.5-3, URL https://CRAN.R-project.org/package=ECOSolveR.

Fu A, Narasimhan B, Boyd S (2017). "**CVXR**: An R Package for Disciplined Convex Optimization." *arXiv preprint arXiv:1711.07582*. URL https://arxiv.org/abs/1711.07582.

Furini F, Traversi E, Belotti P, Frangioni A, Gleixner A, Gould N, Liberti L, Lodi A, Misener R, Mittelmann H, Sahinidis N, Vigerske S, Wiegele A (2017). "QPLIB: A Library of Quadratic Programming Instances." *Technical report*, Optimization Online. Available at Optimization Online, URL http://www.optimization-online.org/DB_HTML/2017/02/5846.html.

Gantz JF, Chute C, Manfrediz A, Minton S, Reinsel D, Schlichting W, Toncheva A (2008). "The Diverse and Exploding Digital Universe: An Updated Forecast of Worldwide Information Growth Through 2011." *IDC White Paper*. URL https://www.calvin.edu/~dsc8/documents/diverse-exploding-digital-universe.pdf.

Gay DM (1985). "Electronic Mail Distribution of Linear Programming Test Problems." *Mathematical Programming Society COAL Newsletter*, **13**, 10–12. URL ftp://lab.unb.br/pub/math/netlib/lp/data/nams.ps.gz.

Geyer CJ, Meeden GD (2019). ***rcdd**: Computational Geometry*. R package version 1.2-2, URL https://CRAN.R-project.org/package=rcdd.

Ghalanos A, Theußl S (2015). ***Rsolnp**: General Non-Linear Optimization Using Augmented Lagrange Multiplier Method*. R package version 1.16.

Ghemawat S, Gobioff H, Leung S (2003). "The Google File System." In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pp. 29–43. ACM Press, New York, NY, USA. doi:https://dx.doi.org/10.1145/1165389.945450.

Gilbert P, Varadhan R (2019). ***numDeriv**: Accurate Numerical Derivatives*. R package version 2016.8-1.1, URL https://CRAN.R-project.org/package=numDeriv.

Goldfarb D, Idnani A (1983). "A Numerically Stable Dual Method for Solving Strictly Convex Quadratic Programs." *Mathematical Programming*, **27**(1), 1–33. ISSN 1436-4646. doi:10.1007/bf02591962.

Gomory RE (1960). "An Algorithm for the Mixed-Integer Problem." *Technical report*, The RAND Corporation. oai: AD0616505.

Grant M, Boyd S (2014). "**CVX**: MATLAB Software for Disciplined Convex Programming, Version 2.1." http://cvxr.com/cvx.

Gregor S, Hevner AR (2013). "Positioning and Presenting Design Science Research for Maximum Impact." *MIS quarterly*, **37**(2), 337–355. URL https://www.jstor.org/stable/43825912.

Griffiths TL, Steyvers M (2004). "Finding Scientific Topics." *Proceedings of the National Academy of Sciences of the United States of America*, **101**, 5228–5235. doi:10.1073/pnas.0307752101.

Gropp W, Moré JJ (1997). "Optimization Environments and the NEOS Server." In MD Buhman, A Iserles (eds.), *Approximation Theory and Optimization*, pp. 167 – 182. Cambridge University Press.

Gurobi Optimization, LLC (2018). "**Gurobi** Optimizer Reference Manual." Version 8.1, URL https://www.gurobi.com.

Harrell Jr FE (2019). ***rms**: Regression Modeling Strategies*. R package version 5.1-3.1, URL https://CRAN.R-project.org/package=rms.

Helmberg C (2000). "Semidefinite Programming for Combinatorial Optimization." URL https://opus4.kobv.de/opus4-zib/files/602/ZR-00-34.pdf.

Helwig NE (2018). ***Dykstra**: Quadratic Programming Using Cyclic Projections*. R package version 1.0-0, URL https://CRAN.R-project.org/package=Dykstra.

Hevner AR, March ST, Park J, Ram S (2004). "Design Science in Information Systems Research." *MIS Quarterly*, **28**(1), 75–105. doi:10.2307/25148625.

Hochreiter R, Schwendinger F (2019). ***ROI.plugin.neos**: NEOS Plug-in for the R Optimization Interface*. R package version 0.3-1, URL https://cran.r-project.org/package=ROI.plugin.neos.

Hocking TD, Joulin A, Bach F, Vert JP (2011). "Clusterpath: An Algorithm for Clustering Using Convex Fusion Penalties." In L Getoor, T Scheffer (eds.),

Proceedings of the 28th International Conference on Machine Learning (ICML-11), ICML '11, pp. 745–752. ACM, New York, NY, USA. ISBN 978-1-4503-0619-5. URL http://citeseerx.ist.psu.edu/viewdoc/versions?doi=10.1.1.228.7220.

Hogg R, Denison D (eds.) (2008). *A History of the English Language*. Cambridge University Press.

Hornik K, Harter R, Theußl S (2017a). ***Rsymphony**: Symphony in R*. R package version 0.1-28, URL https://CRAN.R-project.org/package=Rsymphony.

Hornik K, Meyer D (2007). "Deriving Consensus Rankings from Benchmarking Experiments." In R Decker, HJ Lenz (eds.), *Advances in Data Analysis (Proceedings of the 30th Annual Conference of the Gesellschaft für Klassifikation e.V., Freie Universität Berlin, March 8–10, 2006*, Studies in Classification, Data Analysis, and Knowledge Organization, pp. 163–170. Springer-Verlag.

Hornik K, Meyer D (2010). *relations: Data Structures and Algorithms for Relations*. R package version 0.5-7.

Hornik K, Meyer D, Buchta C (2019). ***slam**: Sparse Lightweight Arrays and Matrices*. R package version 0.1-46, URL https://CRAN.R-project.org/package=slam.

Hornik K, Meyer D, Schwendinger F (2017b). ***ROI.plugin.msbinlp**: Multi-Solution Binary Linear Problem Plug-in for the R Optimization Infrastructure*. R package version 0.3-0, URL https://CRAN.R-project.org/package=ROI.plugin.msbinlp.

Hult GTM, Neese WT, Bashaw RE (1997). "Faculty Perceptions of Marketing Journals." *Journal of Marketing Education*, **19**(1), 37–52.

ILOG I (2019). "**CPLEX** Optimization Studio v12.9.0 Documentation." URL https://www.ibm.com/support/knowledgecenter/SSSA5P_12.9.0.

Johnson SG (2019). "**NLopt** Documentation." URL https://nlopt.readthedocs.io/en/latest/.

Kallrath J (2004). *Mathematical Optimization and the Role of Modeling Languages*, chapter 1, pp. 3–24. Springer-Verlag, Boston, MA. ISBN 978-1-4613-0215-5. doi:10.1007/978-1-4613-0215-5_1.

Karatzoglou A, Smola A, Hornik K (2019). ***kernlab**: Kernel-Based Machine Learning Lab*. R package version 0.9-28, URL https://CRAN.R-project.org/package=kernlab.

Karatzoglou A, Smola A, Hornik K, Zeileis A (2004). "**kernlab** - An S4 Package for Kernel Methods in R." *Journal of Statistical Software*, **11**(1), 1–20. ISSN 1548-7660. doi:10.18637/jss.v011.i09.

Kemeny JG, Snell JL (1962). *Mathematical Models in the Social Sciences*, chapter II. MIT Press, Cambridge.

Koch T, Achterberg T, Andersen E, Bastert O, Berthold T, Bixby RE, Danna E, Gamrath G, Gleixner AM, Heinz S, Lodi A, Mittelmann H, Ralphs T, Salvagnin D, Steffy DE, Wolter K (2011). "MIPLIB 2010." *Mathematical Programming Computation*, **3**(2), 103–163. doi:10.1007/s12532-011-0025-9.

Koecher M (1957). "Positivitätsbereiche Im R^n." *American Journal of Mathematics*, **79**(3), 575–596. ISSN 00029327, 10806377. doi:10.2307/2372563.

Koenker R, Mizera I (2014). "Convex Optimization in R." *Journal of Statistical Software*, **60**(1), 1–23. ISSN 1548-7660. doi:10.18637/jss.v060.i05.

Konis K (2019). ***lpSolveAPI**: R Interface to lp_solve 5.5.2.0*. R package version 5.5.2.0-17.4, URL https://CRAN.R-project.org/package=lpSolveAPI.

Lämmel R (2007). "Google's MapReduce Programming Model—Revisited." *Science of Computer Programming*, **68**(3), 208–237.

Land AH, Doig AG (1960). "An Automatic Method for Solving Discrete Programming Problems." *Econometrica*, **28**(3), 497–520. ISSN 00129682, 14680262. doi:10.2307/1910129.

Lauritzen SL (1996). *Graphical Models*, volume 17. Clarendon Press.

Lewis D (1997). "Reuters-21578 Text Categorization Test Collection." URL http://www.daviddlewis.com/resources/testcollections/reuters21578/.

Lewis DD, Yang Y, Rose TG, Li F (2004). "RCV1: A New Benchmark Collection for Text Categorization Research." *The Journal of Machine Learning Research*, **5**, 361–397.

Liao X, Meyer MC (2014). "**coneproj**: An R Package for the Primal or Dual Cone Projections with Routines for Constrained Regression." *Journal of Statistical Software, Articles*, **61**(12), 1–22. ISSN 1548-7660. doi:10.18637/jss.v061.i12.

Linderoth JT, Ralphs TK (2005). *Noncommercial Software for Mixed-Integer Linear Programming*, chapter 10, pp. 253–303. Operations Research Series. CRC Press. ISBN 978-0-8493-1914-3. doi:10.1201/9781420039597-15.

Lindo Systems I (2013). "**Lindo** User's Manual." URL http://www.lindo.com/.

Lindsten F, Ohlsson H, Ljung L (2011). "Just Relax and Come Clustering!: A Convexification of K-Means Clustering." *Technical Report 2992*, Linköping University, The Institute of Technology. diva: diva2:650707.

Lobo MS, Vandenberghe L, Boyd S, Lebret H (1998). "Applications of Second-Order Cone Programming." *Linear Algebra and its Applications*, **284**(1–3), 193–228. ISSN 0024-3795. doi:10.1016/s0024-3795(98)10032-0.

Lubin M, Dunning I (2015). "Computing in Operations Research Using Julia." *INFORMS Journal on Computing*, **27**(2), 238–248. doi:10.1287/ijoc.2014.0623.

Lumley T (2017). ***leaps**: Regression Subset Selection*. R package version 3.0, URL https://CRAN.R-project.org/package=leaps.

Lumley T, Kronmal R, Ma S (2006). "Relative Risk Regression in Medical Research: Models, Contrasts, Estimators, and Algorithms." *Collection of Biostatistics Research Archive*. URL https://biostats.bepress.com/uwbiostat/paper293.

Luo J, Zhang J, Sun H (2014). "Estimation of Relative Risk Using a Log-Binomial Model with Constraints." *Computational Statistics*, **29**(5), 981–1003. ISSN 1613-9658. doi:10.1007/s00180-013-0476-8.

Luraschi J, Kuo K, Ruiz E (2019). *Mastering **Spark** with R: The Complete Guide to Large-Scale Analysis and Modeling*. O'Reilly Media. URL https://therinspark.com/.

Löfberg J (2004). "**YALMIP**: A Toolbox for Modeling and Optimization in MATLAB." In *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pp. 284–289. IEEE. doi:10.1109/cacsd.2004.1393890.

Mair P, de Leeuw J, Groenen JF (2019). ***smacof**: Multidimensional Scaling*. R package version 2.0-0, URL https://CRAN.R-project.org/package=smacof.

Makhorin A (2011). *GNU Linear Programming Kit Reference Manual Version 4.47*. URL http://www.gnu.org/software/glpk/.

Manning CD, Raghavan P, Schütze H (2008). *An Introduction to Information Retrieval*, volume 1. Cambridge University Press. URL http://nlp.stanford.edu/IR-book/.

March ST, Smith GF (1995). "Design and Natural Science Research on Information Technology." *Decision Support Systems*, **15**(4), 251–266. doi:10.1016/0167-9236(94)00041-2.

Meinshausen N, Bühlmann P (2006). "High-Dimensional Graphs and Variable Selection with the Lasso." *The Annals of Statistics*, **34**(3), 1436–1462.

Message Passing Interface Forum (1994). *MPI: A Message-Passing Interface Standard*.

Message Passing Interface Forum (2003). *MPI-2: Extensions to the Message-Passing Interface*.

Meyer D (2019). ***registry***: *Infrastructure for R Package Registries*. R package version 0.5.1, URL https://CRAN.R-project.org/package=registry.

Meyer D, Hornik K (2019). ***relations***: *Data Structures and Algorithms for Relations*. R package version 0.6-9, URL https://CRAN.R-project.org/package=relations.

Meyer MC, Liao X (2018). ***coneproj***: *Primal or Dual Cone Projections with Routines for Constrained Regression*. R package version 1.14, URL https://cran.r-project.org/package=coneproj.

Michel JB, Shen YK, Aiden AP, Veres A, Gray MK, The Google Books Team, Pickett JP, Hoiberg D, Clancy D, Norvig P, Orwant J, Pinker S, Nowak MA, Aiden EL (2011). "Quantitative Analysis of Culture Using Millions of Digitized Books." *Science*, **331**, 176–182. doi:10.1126/science.1199644.

Miller A (2002). *Subset Selection in Regression*. CRC Press. doi:10.1201/9781420035933.

Miller GA (1995). "WordNet: a Lexical Database for English." *Communications of the ACM*, **38**(11), 39–41. doi:10.1145/219717.219748.

Mingers J, Harzing AW (2007). "Ranking Journals in Business and Management: A Statistical Analysis of the Harzing Data Set." *European Journal of Information Systems*, **16**(4), 303–316.

Mullen K (2014a). "Continuous Global Optimization in R." *Journal of Statistical Software*, **60**(1), 1–45. ISSN 1548-7660. doi:10.18637/jss.v060.i06.

Mullen K (2014b). ***globalOptTests***: *Objective Functions for Benchmarking the Performance of Global Optimization Algorithms*. R package version 1.1, URL https://CRAN.R-project.org/package=globalOptTests.

Mullen K, Ardia D, Gil D, Windover D, Cline J (2011). "**DEoptim**: An R Package for Global Optimization by Differential Evolution." *Journal of Statistical Software, Articles*, **40**(6), 1–26. ISSN 1548-7660. doi:10.18637/jss.v040.i06.

Nash JC (2014a). "On Best Practice Optimization Methods in R." *Journal of Statistical Software*, **60**(1), 1–14. ISSN 1548-7660. doi:10.18637/jss.v060.i02.

Nash JC (2014b). ***Rcgmin**: Conjugate Gradient Minimization of Nonlinear Functions*. R package version 2013-2.21, URL https://CRAN.R-project.org/package=Rcgmin.

Nash JC (2018). ***Rvmmin**: Variable Metric Nonlinear Function Minimization*. R package version 2018-4.17, URL https://CRAN.R-project.org/package=Rvmmin.

Nash JC, Varadhan R (2011). "Unifying Optimization Algorithms to Aid Software System Users: **optimx** for R." *Journal of Statistical Software*, **43**(9), 1–14. ISSN 1548-7660. doi:10.18637/jss.v043.i09.

Nemirovski A (2006). "Advances in Convex Optimization: Conic Programming." In *In Proceedings of International Congress of Mathematicians*, pp. 413–444. ISSN 978-3-03719-522-2. doi:10.4171/022.

Nesterov Y (2004). *Introductory Lectures on Convex Optimization*. Springer-Verlag. doi:10.1007/978-1-4419-8853-9.

Nielsen HB, Mortensen SB (2016). ***ucminf**: General-Purpose Unconstrained Non-Linear Optimization*. R package version 1.1-4, URL https://CRAN.R-project.org/package=ucminf.

Nocedal J, Wright SJ (2006). *Numerical Optimization*. Springer-Verlag. ISBN 978-0387-30303-1. doi:10.1007/978-0-387-40065-5.

O'Donoghue B (2015). "**SCS** - (Splitting Conic Solver)." https://github.com/cvxgrp/scs.git.

O'Donoghue B, Chu E, Parikh N, Boyd S (2016). "Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding." *Journal of Optimization Theory and Applications*, pp. 1–27. ISSN 1573-2878. doi:10.1007/s10957-016-0892-3.

Ormerod JT, Wand MP (2018). ***LowRankQP**: Low Rank Quadratic Programming*. R package version 1.0.3, URL https://CRAN.R-project.org/package=LowRankQP.

Pelckmans K, De Brabanter J, De Moor B, Suykens J (2005). "Convex Clustering Shrinkage." In *Workshop on Statistics and Optimization of Clustering Workshop (PASCAL)*. URL https://lirias.kuleuven.be/handle/123456789/181608.

Perez RE, Jansen PW, Martins JRRA (2012). "**pyOpt**: A Python-Based Object-Oriented Framework for Nonlinear Constrained Optimization." *Structures and Multidisciplinary Optimization*, **45**(1), 101–118. doi:10.1007/s00158-011-0666-3.

Peterson BG, Carl P (2018). ***PortfolioAnalytics**: Portfolio Analysis, Including Numerical Methods for Optimization of Portfolios*. R package version 1.1-0, URL https://cran.r-project.org/package=PortfolioAnalytics.

Pfaff B (2015). ***cccp**: Cone Constrained Convex Problems*. R package version 0.2-4, URL https://CRAN.R-project.org/package=cccp.

Pfaff B (2017). ***rneos**: XML-RPC Interface to NEOS*. R package version 0.3-2, URL https://CRAN.R-project.org/package=rneos.

Pieters R, Baumgartner H, Vermunt J, Bijmolt T (1999). "Importance and Similarity in the Evolving Citation Network of the International Journal of Research in Marketing." *International Journal of Research in Marketing*, **16**(2), 113–128.

Pilato CM, Collins-Sussman B, Fitzpatrick BW (2009). *Version Control with Subversion*. O'Reilly. Full book available online at http://svnbook.red-bean.com/.

Polonsky M, Whitelaw P (2005). "What Are we Measuring When we Evaluate Journals?" *Journal of Marketing Education*, **27**(2), 189–201.

Polonsky MJ (2008). "Publishing on Publishing: Streams in the Literature." *European Business Review*, **20**(5), 401–420.

Porter M (1980). "An Algorithm for Suffix Stripping." *Program*, **3**, 130–137.

Python Software Foundation (2017). *Python Language Reference, Version 2.7*. Wilmington, DE. URL http://www.python.org/.

Racine JS, Nie Z (2018). ***crs**: Categorical Regression Splines*. R package version 0.15-31, URL https://CRAN.R-project.org/package=crs.

Rainer RKJ, Miller MD (2005). "Examining Differences Across Journal Rankings." *Communications of the ACM*, **48**(2), 91–94.

Ralphs TK, Güzelsoy M (2005). *The **SYMPHONY** Callable Library for Mixed Integer Programming*, chapter 2, pp. 61–76. Springer-Verlag. ISBN 978-0-387-23529-5. doi:10.1007/0-387-23529-9_5.

Ralphs TK, Güzelsoy M (2011). ***SYMPHONY** 5.6.14 User's Manual*. Department of Industrial and Systems Engineering, Lehigh University. URL https://www.coin-or.org/SYMPHONY/man-5.6/.

R Core Team (2019a). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

R Core Team (2019b). *Writing R Extensions*. For R, version 3.6.2 (2019-12-12), URL https://cran.r-project.org/doc/manuals/R-exts.html.

Regenwetter M, Rykhlevskaia E (2004). "On the (Numerical) Ranking Associated With any Finite Binary Relation." *Journal of Mathematical Psychology*, (48), 239–246.

Régnier S (1965). "Sur quelques aspects mathématiques des problèmes de classification automatique." *ICC Bulletin*, **4**, 175–191.

Revolution Analytics (2013). ***rhdfs**: R and **Hadoop** Distributed Filesystem*. R package version 1.0.8, URL https://github.com/RevolutionAnalytics/RHadoop/wiki/Downloads.

Revolution Analytics (2015). ***rmr2**: R and **Hadoop** Streaming Connector*. R package version 3.3.1, URL https://github.com/RevolutionAnalytics/RHadoop/wiki/Downloads.

Roettger M, Gelius-Dietrich G, Fritzemeier CJ (2019). ***clpAPI**: R Interface to C API of COIN-OR **Clp***. R package version 1.2.11, URL https://CRAN.R-project.org/package=clpAPI.

Roettger M, Gelius-Dietrich G, Luangkesorn L (2018). ***glpkAPI**: R Interface to C API of **GLPK***. R package version 1.3.1, URL https://CRAN.R-project.org/package=glpkAPI.

Rosenberg DS (2010). ***HadoopStreaming**: Utilities for using R scripts in **Hadoop** streaming*. R package version 0.2, URL http://CRAN.R-project.org/package=HadoopStreaming.

Rosenbrock HH (1960). "An Automatic Method for Finding the Greatest or Least Value of a Function." *The Computer Journal*, **3**(3), 175. /oup/backfile/Content_public/Journal/comjnl/3/3/10.1093/comjnl/3.3.175/2/030175.pdf, URL http://dx.doi.org/10.1093/comjnl/3.3.175.

Rudy J (2011). ***CLSOCP**: A Smoothing Newton Method SOCP Solver*. R package version 1.0, URL https://CRAN.R-project.org/package=CLSOCP.

Sandhaus E (2008). "The New York Times Annotated Corpus." URL http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2008T19.

Schmidberger M, Morgan M, Eddelbuettel D, Yu H, Tierney L, Mansmann U (2009). "State of the Art in Parallel Computing with R." *Journal of Statistical Software*, **31**(1), 1–27. ISSN 1548-7660. URL http://www.jstatsoft.org/v31/i01.

Schrader U, Hennig-Thurau T (2009). "VHB-JOURQUAL2: Method, Results, and Implications of the German Academic Association for Business Research's Journal Ranking." *BuR - Business Research*, **2**(2), 180–204.

Schumacher D (2018). ***ompr**: Model and Solve Mixed Integer Linear Programs*. R package version 0.8.0, URL https://cran.r-project.org/package=ompr.

Schwendinger F (2017). ***ROI.models.globalOptTests**: **ROI** Optimization Problems Based on **globalOptTests***. R package version 1.1, URL https://CRAN.R-project.org/package=ROI.models.globalOptTests.

Schwendinger F (2018). ***ROI.plugin.qpoases**: qpOASES Plugin for the R Optimization Infrastructure*. R package version 0.3-2, URL https://cran.r-project.org/package=ROI.plugin.qpoases.

Schwendinger F (2019). ***ROI.models.netlib**: **ROI** Optimization Problems Based on NETLIB-LP*. R package version 1.1, URL https://CRAN.R-project.org/package=ROI.models.netlib.

Schwendinger F, O'Donoghue B (2019). ***scs**: Splitting Conic Solver*. R package version 1.3-1, URL https://CRAN.R-project.org/package=scs.

Schwendinger F, Theußl S (2019). ***ROI.models.miplib**: R Optimization Infrastructure: MIPLIB 2010 Benchmark Instances*. R package version 0.0-2, URL https://CRAN.R-project.org/package=ROI.models.miplib.

Serrano SA (2015). *Algorithms for Unsymmetric Cone Optimization and an Implementation for Problems with the Exponential Cone*. Ph.D. thesis, Stanford University. URL https://web.stanford.edu/group/SOL/dissertations/ThesisAkleAdobe-augmented.pdf.

Stellato B, Banjac G, Goulart P, Bemporad A, Boyd S (2017). "OSQP: An Operator Splitting Solver for Quadratic Programs." *ArXiv e-prints*. 1711.08013.

Stellato B, Banjac G, Goulart P, Boyd S (2019). ***osqp**: Quadratic Programming Solver Using the OSQP Library*. R package version 0.6.0.3, URL https://CRAN.R-project.org/package=osqp.

Tan KM, Witten D, others (2015). "Statistical Properties of Convex Clustering." *Electronic Journal of Statistics*, **9**(2), 2324–2347. doi:10.1214/15-ejs1074.

Tang J, He G, Dong L, Fang L (2012). "A New One-Step Smoothing Newton Method for Second-Order Cone Programming." *Applications of Mathematics*, **57**(4), 311–331. doi:10.1007/s10492-012-0019-6.

Tellis GJ, Chandy RK, Ackerman DS (1999). "In Search of Diversity: The Record of Major Marketing Journals." *Journal of Marketing Research*, **36**(1), 120–132.

Tetlock P (2007). "Giving Content to Investor Sentiment: The Role of Media in the Stock Market." *The Journal of Finance*, **62**(3), 1139–1168.

The Apache Software Foundation (2019). "Hadoop." Release 3.2.1, URL https://hadoop.apache.org/docs/stable/.

The MathWorks Inc (2017). *MATLAB - The Language of Technical Computing*. Version R2017b, URL https://www.mathworks.com/.

Theoharakis V, Hirst A (2002). "Perceptual Differences of Marketing Journals: A Worldwide Perspective." *Marketing Letters*, **13**(4), 389–402.

Theußl S, Feinerer I (2015). ***tm.plugin.dc**: Text Mining Distributed Corpus Plug-In*. R package version 0.2-8, URL http://CRAN.R-project.org/package=tm.plugin.dc.

Theußl S, Feinerer I (2019). ***hive**: Hadoop InteractiVE*. R package version 0.2-2, URL http://CRAN.R-project.org/package=hive.

Theußl S, Feinerer I (2020). ***DSL**: Distributed Storage and Lists*. R package version 0.1-7, URL http://CRAN.R-project.org/package=DSL.

Theußl S, Feinerer I, Hornik K (2012). "A **tm** Plug-In for Distributed Text Mining in R." *Journal of Statistical Software*, **51**(5), 1–31. doi:10.18637/jss.v051.i05. URL http://www.jstatsoft.org/v51/i05.

Theußl S, Ligges U, Hornik K (2011). "Prospects and Challenges in R Package Development." *Computational Statistics*, **26**(3), 395–404. doi:10.1007/s00180-010-0205-5.

Theußl S, Reutterer T, Hornik K (2014). "How to Derive Consensus Among Various Marketing Journal Rankings?" *Journal of Business Research*, **67**(5), 998–1006. doi:10.1016/j.jbusres.2013.08.006.

Theußl S, Schwendinger F, Hornik K (2019). "ROI: An Extensible R Optimization Infrastructure." *Research Report Series / Department of Statistics and Mathematics 133*, WU Vienna University of Economics and Business, Vienna. URL https://epub.wu.ac.at/5858/.

Theußl S (2017). ***ROI.plugin.glpk**: **ROI** Plug-in **GLPK***. R package version 0.3-0, URL https://CRAN.R-project.org/package=ROI.plugin.glpk.

Theußl S, Borchers HW, Schwendinger F (2019a). "CRAN Task View: Optimization and Mathematical Programming." URL https://CRAN.R-project.org/view=Optimization.

Theußl S, Bravo HC (2016). ***Rcplex**: R Interface to **CPLEX***. R package version 0.3-3, URL https://CRAN.R-project.org/package=Rcplex.

Theußl S, Hornik K (2019). ***Rglpk**: R/GNU Linear Programming Kit Interface*. R package version 0.6-4, URL https://CRAN.R-project.org/package=Rglpk, http://www.gnu.org/software/glpk/.

Theußl S, Schwendinger F, Hornik K, Meyer D (2019b). ***ROI**: R Optimization Infrastructure*. R package version 0.3-2, URL https://CRAN.R-project.org/package=ROI, URL https://ROI.R-Forge.R-project.org/.

Theußl S, Zeileis A (2009). "Collaborative Software Development Using R-Forge." *The R Journal*, **1**(1), 9–14. URL https://journal.R-project.org/archive/2009-1/RJournal_2009-1_Theussl+Zeileis.pdf.

Tibshirani R (1996). "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society B*, **58**(1), 267–288. URL http://www.jstor.org/stable/2346178.

Tierney L, Rossini A, Li N, Sevcikova H (2018). ***snow**: Simple Network of Workstations*. R package version 0.4-3, URL http://CRAN.R-project.org/package=snow.

Tukey JW (1962). "The Future of Data Analysis." *The Annals of Mathematical Statistics*, **33**(1), 1–67.

Turlach BA, Weingessel A (2019). ***quadprog**: Functions to Solve Quadratic Programming Problems*. R package version 1.5-7, URL https://CRAN.R-project.org/package=quadprog.

Udell M, Mohan K, Zeng D, Hong J, Diamond S, Boyd S (2014). "Convex Optimization in Julia." In *Proceedings of the 1st First Workshop for High Performance Technical Computing in Dynamic Languages*, HPTCDL '14, pp. 18–28. IEEE Press, Piscataway, NJ, USA. ISBN 978-1-4799-7020-9. doi:10.1109/hptcdl.2014.5.

Vaishnavi V, Kuechler W, Petter S (2004/19). "Design Science Research in Information Systems." January 20, 2004 (created in 2004 and updated until 2015 by Vaishnavi, V and Kuechler, W); last updated (by

Vaishnavi, V and Petter, S), June 30, 2019, URL http://desrist.org/design-research-in-information-systems/.

van Rijsbergen C (1979). *Information Retrieval*, volume 2. Butterworths. URL http://www.dcs.gla.ac.uk/Keith/Preface.html.

Vandenberghe L, Boyd S (1996). "Semidefinite Programming." *SIAM Review*, **38**(1), 49–95. doi:10.1137/1038003. https://doi.org/10.1137/1038003.

Varadhan R (2015). ***alabama**: Constrained Nonlinear Optimization*. R package version 2015.3-1, URL https://CRAN.R-project.org/package=alabama.

Varadhan R, Gilbert P (2009). "**BB**: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function." *Journal of Statistical Software, Articles*, **32**(4), 1–26. ISSN 1548-7660. doi:10.18637/jss.v032.i04.

Wakabayashi Y (1998). "The Complexity of Computing Medians of Relations." *Resenhas*, **3**(3), 323–349.

Würtz D (2014). ***Rnlminb2**: An R Extension Library for Constrained Optimization with Nlminb*. R package version 3002.10, URL https://r-forge.r-project.org/projects/rmetrics/.

Würtz D (2017). ***Rdonlp2**: An R Extension Library to Use Peter Spelluci's **DONLP2** from R*. R package version 3042.11, URL https://r-forge.r-project.org/projects/rmetrics/.

Wächter A, Biegler LT (2006). "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming." *Mathematical Programming*, **106**(1), 25–57. ISSN 1436-4646. doi:10.1007/s10107-004-0559-y.

Ypma J (2011). ***ipoptr**: R Interface to **Ipopt***. R package version 0.8.4, URL http://r-forge.r-project.org/projects/ipoptr/.

Ypma J, Borchers HW, Eddelbuettel D (2018). ***nloptr**: R Interface to **NLopt***. R package version 1.2.1, URL https://CRAN.R-project.org/package=nloptr.

Yu H (2002). "**Rmpi**: Parallel Statistical Computing in R." *R News*, **2**(2), 10–14. URL http://www.r-project.org/doc/Rnews/Rnews_2002-2.pdf.

Yu H (2018). ***Rmpi**: Interface (Wrapper) to MPI (Message-Passing Interface)*. R package version 0.6-9, URL https://CRAN.R-project.org/package=Rmpi.

Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X, Rosen J, Venkataraman S, Franklin MJ, *et al.* (2016). "Apache Spark: a Unified Engine for Big Data Processing." *Communications of the ACM*, **59**(11), 56–65. doi:10.1145/2934664.

Zhou D, Ma J, Turban E (2001). "Journal Quality Assessment: An Integrated Subjective and Objective Approach." *IEEE Transactions on Engineering Management*, **48**(4), 479–490.

Zhu C, Xu H, Leng C, Yan S (2014). "Convex Optimization Procedure for Clustering: Theoretical Revisit." In Z Ghahramani, M Welling, C Cortes, N d Lawrence, K q Weinberger (eds.), *Advances in Neural Information Processing Systems 27*, pp. 1619–1627. Curran Associates, Inc. URL http://papers.nips.cc/paper/5307-convex-optimization-procedure-for-clustering-theoretical-revisit.pdf.

Zhu Z, Ye Y (2016). ***Rdsdp**: R Interface to the **DSDP** Semidefinite Programming Library*. R package version 1.0.4-2, URL https://CRAN.R-project.org/package=Rdsdp.